MASTER EN SCIENCES INFORMATIQUES

Inversion automatique de programmes COBOL

Henry, Benoît

Award date:
1988

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
INSTITUT D'INFORMATIQUE
NAMUR

INVERSION AUTOMATIQUE DE PROGRAMMES COBOL

PROMOTEUR : B. LECHARLIER

Mémoire présenté par
Benoît HENRY en vue de
l'obtention du titre
de Licencié et Maître
en Informatique

ANNEE ACADEMIQUE 1987-1988

Je tiens à remercier Mr B. Lecharlier , qui m'a fourni ce sujet, pour l'aide précieuse qu'il m'a apportée durant réalisation de ce mémoire.

RESUME.

Le but de ce mémoire est de concevoir et d'implémenter un logiciel qui réalisera l'inversion automatique de programmes COBOL. Ce logiciel est décomposé en quatre parties : un analyseur lexical, un programme qui supprime les instructions "PERFORM", un programme qui effectue l'inversion et enfin un programme qui reconstruit le texte COBOL. Après avoir introduit la notion d'inversion, nous décrivons les règles de l'inversion, les spécifications de ces programmes, puis nous détaillons la méthode d'implémentation. Enfin, nous présentons un gestionnaire qui permet d'effectuer des inversions en chaîne de programmes.

ABSTRACT.

The dissertation's aim is the conception and programmation of a package that realize Cobol program inversion. The package is composed of 4 parts : a lexical analyser, a perform free transformer routine, a program inversion routine and a constructer routine. First, we describe inversion rules and specifications of those routines. Then, the implementation of those routines is discussed. At end, we show a package that allow to perform multi-inversion.

TABLE DES MATIERES

CHAPITRE I. INTRODUCTION.

1.1 NOTION D'INVERSION.

L'inversion permet de simuler le mécanisme de co-routine tel que défini par KNUTH dans "The art of computer programming, Vol 1".

concept de co-routine :

Une co-routine peut être comparée à une sous-routine, à la différence suivante : lors d'un appel, l'exécution d'une sous-routine débute toujours à son début, c'est-à-dire à une place fixe tandis que, lors de l'appel, l'exécution de la co-routine se poursuit à l'instruction qui suit la dernière instruction exécutée.

Exemple :

- exécution d'une sous-routine

```
 A (principal)          B (sous-routine)
...                    DEBUT
CALL B                 ...
...                    ...
...                    ...
CALL B                 EXIT
```

- exécution de co-routines

```
 A                      B
...                    ...
JMP B                  ...
...                    JMP A
...                    ...
JMP B                  ...
...                    JMP A
```

Cette notion de co-routine n'existe pas en Cobol, et l'inversion permettra donc de la simuler.

La notion d'inversion de programme à été introduite par M JACKSON dans "Principles of program design".

L'inversion consiste à modifier 2 programmes reliés par un flux de données intermédiaire (un fichier séquentiel) dans le but de supprimer ce fichier.

La situation de départ doit être la suivante :
- 2 programmes, soit P1 et P2,
- un fichier intermédiaire séquentiel C entre ces 2 programmes,
sachant que le fichier C est le seul lien entre les programmes P1 et P2.

Les instructions d'écriture dans C de P1 seront remplacées par des appels à P2 et les instructions de lecture dans c de P2 seront remplacées par des appels à P1. Le fichier intermédiaire va donc disparaître.

Pour réaliser cela, 2 possibilités nous sont offertes :
- soit inverser P1, qui deviendra un sous-programme de P2. P2, au lieu de lire un enregistrement du fichier intermédiaire, va appeler P1 qui lui fournira cet enregistrement.
- soit inverser P2, qui deviendra un sous-programme de P1. P1, au lieu d'écrire un enregistrement dans le fichier intermédiaire, va appeler P2 et lui fournira cet enregistrement. P2 traitera cet enregistrement et rendra la main à P1.

Le mécanisme de l'inversion sera expliqué au début du chapître II.

1.2 AVANTAGES DE L'INVERSION.

L'inversion permettra un gain de place : le fichier intermédiaire disparait.

L'inversion procurera aussi un gain de temps : les opérations de lecture et écriture dans le fichier intermédiaire disparaissent et sont remplacées par des instructions "CALL" d'appel à un sous-programme. Or, le temps d'accès à un fichier est nettement plus important que le temps d'exécution d'une instruction Call.

L'inversion permettra enfin d'obtenir rapidement les premiers résultats de P2, avant la fin de P1. Cela peut être utile dans certains cas.

1.3 CHAMPS D'APPLICATION DE L'INVERSION.

Lors de la programmation d'applications concernant le traitement des données, et lorsque les structures de données de départ et de résultat sont différentes, M Jackson conseille de séparer l'application en 2 composants, séparés entre eux par un fichier séquentiel (voir 1.4 exemple). Il propose alors d'inverser manuellement ces programmes.
Cette inversion automatique permettra à ceux qui programment selon la méthode préconisée par M jackson d' éviter d'effectuer manuellement les mécanismes de l'inversion, ce qui est une tâche relativement fastidieuse et répétitive.

Elle permettra aussi à cette personne d'employer les instructions PERFORM, ce qui est nettement plus pratique pour construire un programme lisible et qui était impossible avec la méthode d'inversion de M Jackson.

Pour des applications compliquées, il est général ement plus facile de construire un algorithme en plusieurs passes que un algorithme en une passe. En séparant le traitement en une succession de plus petites étapes, un problème est généralement plus facile à comprendre et fournit une découpe naturelle du travail dans le cas d'une réalisation en équipe.

L'inversion automatique permettra de rassembler les différentes passes de cet algorithme.
Enfin, cette inversion automatique pourra aussi inverser des programmes existants séparés entre eux par un fichier séquentiel.

1.4 EXEMPLE : ANALYSE DE TELEGRAMMES.

Cet exemple est tiré de "principles of program design, M JACKSON", et montre un cas où l'inversion peut être appliquée.
Un fichier en entrée sur bande contient les textes de plusieurs télégrammes. Ce fichier est accessible par une instruction "read block", qui lit un string de longueur variable, délimité par le caractère terminal EOB. La taille d'un block ne peut dépasser 100 caractères, EOB inclus. Chaque bloc contient un certain nombre de mots, séparés par un ou plusieurs espaces. Chaque télégramme est composé de mots et terminé par le mot spécial "ZZZ". Le fichier est terminé par un bloc de fin spécial dont le premier caractère est EOF. De plus, il y a toujours un télégramme vide à la fin du fichier, dans le bloc qui précède le bloc spécial de fin de fichier. Ce télégramme vide est constitué du mot "ZZZ". Il n'y a pas de relations particulières entre les blocs et les télégrammes : un télégramme peut débuter et se terminer n'importe où dans un bloc et se dérouler sur plusieurs blocs. Plusieurs télégrammes peuvent se trouver dans un seul bloc.
Le traitement sera l'analyse de ces télégrammes. Un rapport devra être produit montrant, pour chaque télégramme, le nombre de mots qu'il contient et le nombre de ces mots dont la taille dépasse 12 caractères. Dans ce rapport, les mots "ZZZ" ne comptent pas pour des mots, de même que le télégramme vide ne compte pour un télégramme.
Le format de ce rapport sera donc :

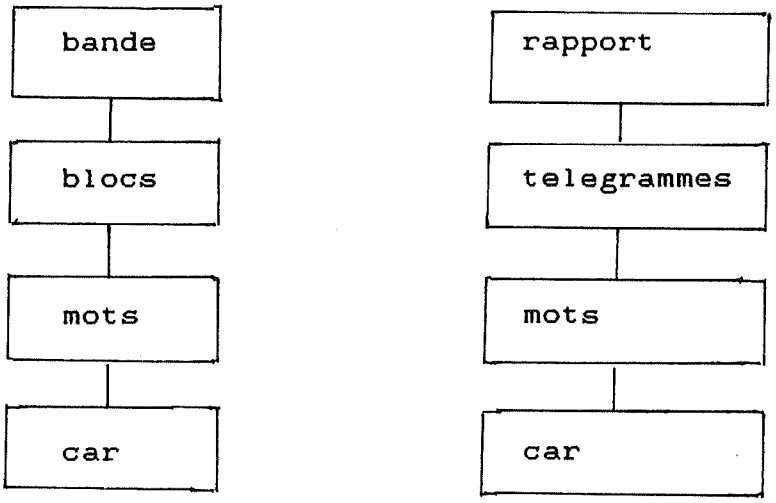
```
ANALYSE DE TELEGRAMMES.
TELEGRAMME 1
    15 MOTS DONT 3 TROP LONGS
TELEGRAMME 2
```

106 MOTS DONT 12 TROP LONGS

....

FIN DE L'ANALYSE.

Les structures de données simplifiées du fichier en entrée et de ce rapport sont donc les suivantes :



Il n'y a pas de correspondance directe entre ces 2 structures. Il y a une correspondance entre le fichier de départ et le rapport : le rapport est dérivé du fichier en entrée. Mais il y a conflit de frontière entre les blocs et les télégrammes du rapport : les mots des blocs sont les mots des télégrammes, et apparaissent dans le même ordre, mais les blocs et les télégrammes eux-mêmes ne correspondent pas entre eux.

Lorsque nous écrirons le programme, nous aurons besoin d'une instruction "read block". Cette instruction fera partie d'un module qui sera exécuté une fois par bloc. Nous aurons aussi une instruction du style "add 1 to telegrams", qui fera partie d'un module exécuté une fois par télégramme. Donc, notre structure de programme devra contenir un composant qui sera exécuté une fois par bloc et un autre qui sera exécuté une fois par télégramme. Ces 2 composants ne peuvent apparaître ensemble dans une seule structure, et nous en aurons donc au moins deux.

La structure du système sera donc la suivante :

P1 créera un fichier intermédiaire à partir du fichier de donnée sur bande; P2 produira le rapport à partir du fichier intermédiaire. Le but est donc de surmonter le conflit de structure et donc s'assurer que seul P1 aura besoin du composant TRAITEMENT D'UN BLOC et que seul P2 aura besoin du composant TRAITEMENT D'UN TELEGRAMME. Que devra donc être le fichier intermédiaire? Il devra pouvoir être construit sans savoir ce qu'est un télégramme (car P1 ignore la notion de télégramme) et devra permettre de construire le rapport par un composant de programme qui ignore la notion de bloc (car P2 ignore la notion de bloc).
Lorsque nous examinons les 2 structures de données simplifiées, nous remarquons qu'il y a une bonne correspondance entre caque paire de composants d'un même niveau, sauf pour la paire bloc-télégramme. Nous pouvons donc résoudre le conflit de structure en choisissant d'écrire comme un simple enregistrement du fichier intermédiaire soit :

- le fichier sur bande en entier
- un mot
- un caractère.

La première solution est irréalisable car cela supposerait de charger tous le fichier en mémoire centrale, en remplaçant chaque EOB par un espace.
La meilleur solution est d'écrire comme enregistrement chaque mot du fichier de départ. C'est la meilleur solution car le mot est le composant le plus grand qui corresponde dans les 2 structures et puisse tenir dans la mémoire centrale.
Si nous avions choisi d'écrire un enregistrement par caractère, nous aurions du disséquer inutilement les mots en caractères dans P1 pour les reconstituer dans P2.
Il semble raisonnable d'inclure à chaque mot un compteur contenant le nombre de caractères du mot. Le format de

chaque enregistrement du fichier intermédiaire sera donc :

```
01 MOT.
   02 COMPT PIC 999.
   02 CORPS-MOT.
      03 CAR-MOT  PIC X OCCURS 100.
```

Cependant, si cette solution résout bien le conflit de structure, elle a aussi de gros inconvénients : la constitution d'un fichier intermédiaire, qui peut être long, le temps d'accès qui en résulte (une opération de lecture et d'écriture par enregistrement) et le fait qu'il faut attendre l'exécution complète de P1 avant d'avoir le début du résultat de P2.

Il existe cependant une solution qui permet de remédier à ces inconvénients : convertir un des 2 programmes P1 ou P2 en un sous-programme de l'autre. Cette conversion sera appelée inversion de programme.
deux cas sont possibles :
- soit inverser P1, qui deviendra un sous-programme de P2. P2, au lieu de lire un enregistrement du fichier intermédiaire, va appeler P1 qui lui fournira cet enregistrement.
- soit inverser P2, qui deviendra un sous-programme de P1. P1, au lieu d'écrire un enregistrement dans le fichier intermédiaire, va appeler P2 et lui fournira cet enregistrement. P2 traitera cet enregistrement et rendra la main à P1.

Les mécanismes de l'inversion seront décrits au début du chapitre 2.

1.5

Le but de ce mémoire est de concevoir et d'implémenter un logiciel qui réalisera l'inversion automatique de programmes Cobol.
Ce logiciel sera décomposé en quatre parties (ou sous-programmes):

- un analyseur lexical décomposera le texte Cobol en mots et facilitera ainsi son exploitation par les autres sous-programmes.
- un re-constructeur recomposera le texte Cobol à partir d'une suite de mots (opération inverse de l'analyseur).
- un programme d'inversion, qui réalisera l'inversion de programmes Cobol.
- un programme de suppression des PERFORM, qui devra être exécuté avant l'inversion.

Dans le chapître II, nous expliquerons les règles formelles de l'inversion et décrirons les spécifications de ces programmes.
Le chapître III détaillera la méthode utilisée pour l'implémentation de ces programmes.
Enfin, le chapître IV présentera un gestionnaire d'inversion qui permettra d'effectuer des inversions en chaîne de plusieurs programmes.

CHAPITRE II : REGLES FORMELLES.

2.1 REGLES FORMELLES DE L'INVERSION.

2.1.1 Mécanisme de l'inversion.

Supposons donc que nous ayons 2 programmes, reliés entre eux par un fichier séquentiel intermédiaire.
Pour pouvoir inverser ces programmes, nous devons connaitre :

- le nom du premier programme : soit P1
- le nom interne du fichier intermédiaire dans le programme P1, soit FICHC
- le nom du deuxième programme : soit P2
- le nom interne du fichier intermédiaire dans le programme P2, soit FICHC.

Notons que les noms internes du fichier dans les programmes P1 et P2 ne sont pas nécessairement identiques; seule leur structure doit être semblable.

Pour supprimer le fichier intermédiaire, 2 solutions sont possibles.

- soit inverser P1 par rapport à P2, dans ce cas, P1 devient un sous-programme de P2, qui, au lieu de lire chaque enregistrement de FICHC séquentiellement, va appeler P1 qui lui fournira cet enregistrement.

- soit inverser P2 par rapport à P1, dans ce cas, P2 devient un sous-programme de P1, qui, au lieu d'écrire chaque enregistrement dans FICHC séquentiellement, va appeler P1 qui traitera cet enregistrement.

Avant d'énoncer les règles formelles de l'inversion, voyons sur un exemple les modifications à apporter aux 2 programmes.

L'execution de P1 et de P2 donnera cette structure :

```
P1                          P2

DEBUT.                      DEBUT.
   DO X                        DO S
   OPEN F                      OPEN F
   DO X1                       DO T
   WRITE F                     READ F AT END instructions
   ...                         ...
   WRITE F                     READ F AT END instructions
   ...                         ...
   WRITE F                     READ F AT END instructions
   DO Y                        DO U
   CLOSE F                     CLOSE F
   DO Z                        DO V
FIN.                        FIN.
```

Etudions le cas de l'inversion de P1 par rapport à P2. P1 deviendra donc un sous-programme de P2.

Les transformations à apporter à P2 sont les suivantes:

- supprimer OPEN F et CLOSE F , qui sont devenus inutile du fait de la disparition du fichier F.

- remplacer chaque instruction READ F par un appel au sous-programme P1-1 , avec comme paramètre l'enregistrement du fichier F :

soit REC, l'enregistrement de F, nous aurons donc :

```
CALL P1-1 USING REC.
```

Il faut également prévoir une variable indiquant la fin du fichier F. Soit EOF,cette variable qui sera remplie par le programme P1-1 et sera aussi passée comme paramètre lors du CALL.
Le AT END éventuel sera remplacé par "IF EOF instructions."
Chaque instruction "READ F AT END instructions" sera donc remplacée par la séquence : "CALL P1-1 USING REC EOF. IF EOF instructions."

Les transformations à apporter à P1 sont les suivantes:

- supprimer OPEN F et CLOSE F , qui sont devenus inutile du fait de la disparition du fichier F.

- Le problème est de repérer chaque point de reprise de l'execution de P1-1. Nous allons donc introduire une variable entière QS, initialisée à 1 dans la WORKING STORAGE SECTION.
Il faudra numéroter les instructions WRITE F de 2 à N+1, et ajouter au début du programme la séquence suivante :

```
     GO TO Q1 Q2   ...   QN+1 DEPENDING ON QS.
   Q1.
```

avec N = le nombre de WRITE F dans le programme.
Si QS est initialisé à 1, lors de la première exécution, QS vaut 1 et l'execution se poursuivra en Q1, c'est-à-dire au debut du programme. Lors des reprises suivantes, l'execution se poursuivra

au point QI, c'est-à-dire au point précédent l'instuction qui suit le WRITE F.

Pour I variant de 2 à N+1, remplacer la Ieme instruction WRITE F par

```
     MOVE I TO QS
     MOVE O TO EOF
     EXIT.
  QI.
```

Enfin, il faudra remplacer le "STOP RUN" par

```
      MOVE 1 TO EOF
      EXIT
```

Ceci permettra au programme P2 de connaitre la fin du fichier F.

Notons que si P2 détecte la fin de fichier autrement que par une clause AT END (par exemple grace à un dernier enregistrement fictif, comme dans l'exemple de l'analyse de télégrammes), ou si P2 ne lit pas complètement P1, le programme P1 ne s'exécutera plus complètement (les instructions DO Y et DO Z se seront pas exécutées).
Dans ce cas, c'est-à-dire lorsque, à la fin de P2, EOF est faux, il faudra donc effectuer une boucle : tant que EOF est faux, CALL P1, juste avant le STOP RUN.
A la place du STOP RUN dans P2, nous aurons donc :

```
     PERFORM APPEL-P1 UNTIL (EOF = 1).
     STOP RUN.
APPEL-P1.
     CALL P1-1 USING REC EOF.
FIN.
```

Voici ce que cela donne sur l'exemple :

P1MOINS1 P2'

avec QS initialisé à 1 dans la WWS.

```
DEBUT.                          DEBUT.
   GO TO Q1, Q2, Q3, Q4
      DEPENDING ON QS.
Q1.
   DO X                            DO S

   DO X1                           DO T
   MOVE 0 TO EOF                   CALL P1MOINS1 USING
   MOVE 2 TO QS                    EOF, RECC
   EXIT                            IF EOF instructions
Q2.
   ...                             ...
   WRITE F                         READ F
    MOVE 0 TO EOF                  CALL P1MOINS1 USING
   MOVE 3 TO QS                    EOF, RECC
   EXIT                            IF EOF instructions
Q3.
   ...                             ...
   WRITE F                         READ F
   MOVE 0 TO EOF                   CALL P1MOINS1 USING
   MOVE 4 TO QS                    EOF, RECC
   EXIT                            IF EOF instructions
Q4.
    DO Y                           DO U
                                   PERFORM APPEL-P1
                                   UNTIL EOF = 1
                                       DO Z

DO V
FIN.                            FIN.
    MOVE 1 TO EOF
    EXIT.

                                APPEL-P1.
                                   CALL P1MOINS1 USING
                                   EOF, RECC
```

Voyons maintenant les règles formelles :

2.1.2. Inversion de P1 par rapport à P2.

P1 va donc devenir un sous-programme de P2 qui, au lieu de lire un enregistrement du fichier intermédiaire C, va appeler directement le programme P1, lequel lui fournira cet enregistrement.

2.1.2.1. Modifications à apporter à P2.

- Dans l'"ENVIRONMENT DIVISION",
  "INPUT-OUTPUT SECTION",
  "FILE CONTROL",
  il faudra supprimer : SELECT fichc ASSIGN TO etc .
  (supprimer jusqu'au point)

- Dans la "DATA DIVISION",
  "FILE SECTION",
  il faudra supprimer la définition du fichier "fichc" et faire passer la déclaration de ses enregistrements dans la WSS.

  "WORKING-STORAGE SECTION"
  ajouter la déclaration de EOF (01 EOF PIC X.)
  et des enregistrements de "fichc" au début de la WWS.

- dans la "PROCEDURE DIVISION",

1) supprimer "OPEN INPUT fichc"

2) remplacer "READ fichc RECORD [INTO rec1] AT END instructions".
avec "instructions" = tout ce qui se trouve entre "AT END" et le point par :

```
CALL p1moins1 USING EOF, (recc)
                         rec1
IF EOF instructions .
```

3) remplacer le "CLOSE fichc" par "PERFORM APPEL-P1 UNTIL EOF = 1"

Ceci pour permettre l'execution complète de P1 lorsque le programme P2 ne lit pas le EOF.

4) apres le STOP RUN, insérer :

```
APPEL-P1.
   CALL p1moins1 USING EOF, rec.
```

2.1.2.2 Modifications à apporter à P1.

- Dans "IDENTIFICATION DIVISION",
  changer le nom du programme P1 et mettre P1-1

- Dans l'"ENVIRONMENT DIVISION",
  "INPUT-OUTPUT SECTION",
  "FILE CONTROL",
  il faudra supprimer : SELECT fichc ASSIGN TO etc .

- Dans la "DATA DIVISION",
  "FILE SECTION",
  supprimer : FD fichc DATA RECORDS ARE recc
  01 recc
  declaration-de-recc.

- introduire une "LINKAGE SECTION"
  avec - la description cobol de recc
  01 recc
  declaration-de-decc
  - la description cobol de EOF
  01 EOF PIC 9.

- Ajouter en "WORKING-STORAGE SECTION"
  une variable entière QS initialisée à 1.

- dans la "PROCEDURE DIVISION",

- numéroter les instructions "WRITE recc [FROM rec1] de 2 à M+1

- supprimer OPEN OUTPUT FICHC.

- à la place de "PROCEDURE DIVISION.", nous aurons :

```
PROCEDURE DIVISION USING EOF, recc.
   GO TO Q1, Q2, ... , QM+1 DEPENDING ON QS.
Q1.
```

avec M = le nombre d'instructions "WRITE recc".

- remplacer la Ième instruction "WRITE recc[FROM rec1]

par :

```
MOVE I TO QS
MOVE O TO EOF
[ MOVE rec1 TO recc ]
SORTIEI.
   EXIT PROGRAM.
QI.
```

Notons que en Cobol, EXIT PROGRAM doit être la seule instruction d'un paragraphe. C'est pour cela que nous avons mis un le lable SORTIEI.

- supprimer CLOSE FICHC.

- remplacer " STOP RUN"

par :

```
MOVE 1 TO EOF
EXIT.
```

2.1.3. Inversion de P2 par rapport à P1.

2.1.3.1 Modifications à apporter à P1.

- Dans l'"ENVIRONMENT DIVISION",
  "INPUT-OUTPUT SECTION",
  "FILE CONTROL",

il faudra supprimer : SELECT fichc ASSIGN TO etc .

- Dans la "DATA DIVISION",
  "FILE SECTION",

il faudra supprimer la définition du fichier "fichc" et faire passer la déclaration de "recc" dans la WSS.

"WORKING-STORAGE SECTION"

ajouter la déclaration de "recc" au début de cette WWS.

- dans la "PROCEDURE DIVISION",

1- remplacer "OPEN OUTPUT fichc"
par CALL p2mois1 USING EOF, recc
cet appel ne transmettra rien et a pour seul but de faire avancer P2moins1 jusqu'au premier "READ".

2- remplacer "WRITE recc [FROM rec1]
par :
MOVE 0 TO EOF
CALL p2moins1 USING EOF, (recc)
rec1

3- remplacer "CLOSE fichc"
par : MOVE 1 TO EOF
CALL p2moins1 USING EOF, recc

2.1.3.2 Modifications à apporter à P2.

- Dans "IDENTIFICATION DIVISION",
changer le nom du programme P2 et mettre P2moins1

- Dans l'"ENVIRONMENT DIVISION",
"INPUT-OUTPUT SECTION",
"FILE CONTROL",
il faudra supprimer : SELECT fichc ASSIGN TO etc .

- Dans la "DATA DIVISION",
"FILE SECTION",
supprimer : FD fichc DATA RECORDS ARE recc
01 recc
declaration-de-recc

- introduire une "LINKAGE SECTION"
avec - la description cobol de recc
01 recc
declaration-de-decc
- la description cobol de EOF

- Ajouter en "WORKING-STORAGE SECTION"
  une variable entière QS initialisée à 1.

- dans la "PROCEDURE DIVISION",
  numéroter les instructions "READ fichc RECORD [INTO rec1] de 2 à M+1

  1) à la place de "PROCEDURE DIVISION.", nous aurons :

```
PROCEDURE DIVISION USING EOF, recc.
      GO TO Q1 Q2 ... QM+1 QM+2 DEPENDING ON QS.
Q1.
```

  2) supprimer OPEN INPUT FICHC.

  3) remplacer la Ième instruction "READ fichc RECORD [INTO rec1] AT END instructions ."
  avec "instructions" = tout ce qui se trouve entre "AT END" et le point

```
par :  MOVE I TO QS
       SORTIEI.
          EXIT PROGRAM.
       QI.
        [ MOVE rec1 TO recc ]
          IF EOF instructions.
```

  4) supprimer "CLOSE fichc"

  5) remplacer le STOP RUN par :

```
     MOVE M+2 TO QS
  Q(M+2).
     EXIT PROGRAM.
```

  Lorsque le programme P2 ne lit pas entièrement le fichier intermédiaire, P1 effectuera donc 1 ou plusieurs appels à P2moins1 quand celui-ci aurra terminé son exécution. Ces instructions permettrons donc à P2moins1 de rendre la main à P1.

2.2 LA TRANSFORMATION DES "PERFORM".

Remarque : Pour éviter de compliquer l'analyseur lexical, les conditions devront se trouver entre parenthèses.

2.2.1. Représentation métalinguistique.

Les mots en majuscule sont des mots du programme COBOL.
Les mots en minuscule sont des variables contenant du texte COBOL.
Les crochets [] délimitent une option facultative.
Une barre verticale | indique une concaténation.
Une paire d'accolades {} indique une alternative.
Une paire << >> indique une répétition
Un nom suivit de * signifie ce nom après transformation.

2.2.2 Description du problème.

Nous avons vu au chapitre précédent que les instructions READ/WRITE étaient remplacées par une instruction CALL dans le programme appelant.
Cependant, en Cobol, si cette instruction CALL se trouve dans un paragraphe appelé par un PERFORM, rien ne dit que le retour s'effectuera encore correctement. En effet, l'adresse de retour pourrait avoir été modifiée durant l'exécution du sous-programme (règle Cobol).

Pour s'assurer que le programme inversé fonctionne correctement, il faut donc que le niveau de pile des PERFORM soit à 0 lors d'une instruction READ/WRITE.
La manière la plus simple et la plus sûre d'arriver à ce résultat est de supprimer tous les PERFORM des programmes et de les remplacer par des instructions équivalentes.

Le PERFORM sera remplacé par 2 séries d'instructions de rôle différent:

- à la place du PERFORM, il faudra un branchement (GO TO) vers le début de la procédure appelée et une adresse de retour ;de plus, si ce n'est pas un PERFORM simple, il faudra d'autres instructions pour le réduire en un PERFORM simple.
- à la fin de la procédure appelée, il faudra organiser le retour vers l'instruction qui suit le PERFORM.

De plus, si nous remplaçons un PERFORM par un branchement et une adresse de retour, et que ce perform se trouve dans une structure IF, la logique du programme sera changée.

exemple :

```
    IF A = 1
       PERFORM TOTO
    ELSE
       MOVE 1 TO A.
```

deviendrait :

```
    IF A = 1
       instr
       GO TO TOTO
 RETOUR-TOTO.
    ELSE
       MOVE 1 TO A.
```

IL faudra donc également transformer les structures IF, de manière à "sortir" les instructions qui s'y trouvent. La manière de le faire sera détaillée au point 4 (transformation des IF).

Nous allons maintenant montrer les règles de transformation du programme pour supprimer les PERFORM et transformer les IF.

SOURCE OBJET

```
prog ::= identification
         environment
         data
         procedure
```

```
prog* ::= identification
          environment
          data*
          procedure*
```

```
data ::= file section
  working-storage section
  linkage section
  communication section
  report section
```

```
data* ::= file section
  working-storage section*
  linkage section
  communication section
  report section
```

```
working-storage section ::=
  déclarations-de-variables
```

```
working-storage section* ::=
  nouvelles-déclarations
  déclarations-de-variables
```

les " nouvelles-declarations " seront définies dans le point 3 (le déroutement).

```
procedure ::=
   << section >>
```

```
procedure* ::=
   << section* >>
```

```
section ::=
   << paragraphe >>
```

```
section* ::=
   << paragraphe* >>
    instructions-de-retour
```

```
paragraphe ::=
 titre-de procedure.
   << instruction >>
```

```
paragraphe* ::=
  titre-de procedure.
    << instruction* >>
  instructions-de-retour
```

les "instructions-de-retour" seront définies dans le point 4 (le retour).

```
instruction ::=

 appel-de-procedure
       ou
 structure-alternative
```

```
instruction* ::=

   appel-de-procedure*
         ou
   structure-alternative*
```

ou
autre-instruction

ou
autre-instruction

"appel de procedure" et "appel de procedure*" seront définis dans le point 3 (le déroutement).

"structure-alternative" et "structure-alternative*" seront définis dans le point 4 (transformation des IF).

2.2.3. La transformation des PERFORMs.

2.2.3.1. Le déroutement.

2.2.3.1.1 Le perform simple.

format de départ :

```
appel-de-procedure ::=   PERFORM proc1
                       [ THRU proc2 ]
```

Règles de transformation :

- soit proc = proc2 si on a l'option : THRU proc2
  proc1 si on a pas cette option.

- soit nbre-appel, le nombre courant d'appel à proc à cet endroit du programme.

- Il faut créer une adresse de retour qui doit être identifiante de cet appel-ci. Je vais donc la créer par exemple en concaténant RET (qui veut dire retour), proc (le nom de la procédure) et nbre-appel (le nombre de fois que l'on à déjà appelé proc).

- Je vais également créer une variable, qui servira pour le retour et qui indiquera de quel appel à la même procédure on est parti.

Le nom de cette variable dépendra donc uniquement de proc.
elle sera la concaténation de VAR et proc.
La valeur contenue dans cette variable sera nbre-appel.
Cette variable sera déclarée au début de la WORKING-STORAGE SECTION.

Format résultat :

```
appel-de-procedure* ::=
                        MOVE nbre-appel TO VAR|proc
                        GO TO proc1.
                    RET|proc|nbre-appel.

nouvelles-déclarations ::=
                          nouvelles-déclarations
                          01 VAR|proc  PIC 99.
```

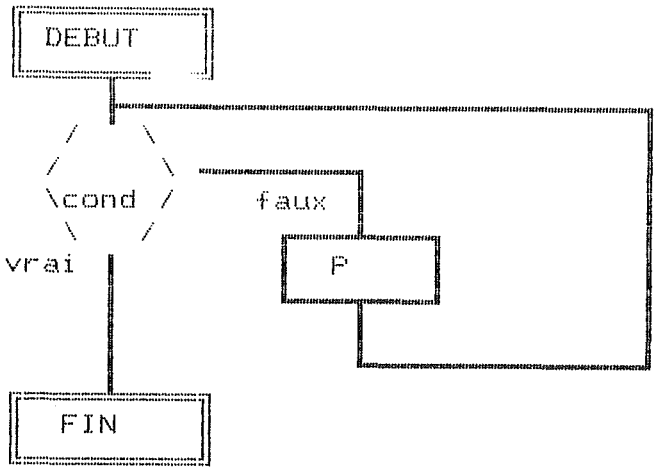
2.2.3.1.2 Le PERFORM option UNTIL.

format de départ :

```
appel-de-procedure ::=
                        PERFORM proc1
                        [ THRU proc2 ]
                        UNTIL (condition)
```

Rappelons que la condition doit être délimitée par des parenthèses.

organigramme :



Règles de transformation :

Les règles de transformations sont les mêmes que pour le perform simple, et il suffit d'ajouter un test après l'adresse de retour et avant de faire le branchement vers la procédure.

format résultat :

```
appel-de-procedure* ::=
                        MOVE nbre-appel TO VAR|proc.
                    RET|proc|nbre-appel.
                        IF NOT condition
                        GO TO proc1


nouvelles-déclarations ::=
                        nouvelles-déclarations
                        01 VAR|proc  PIC 99.
```

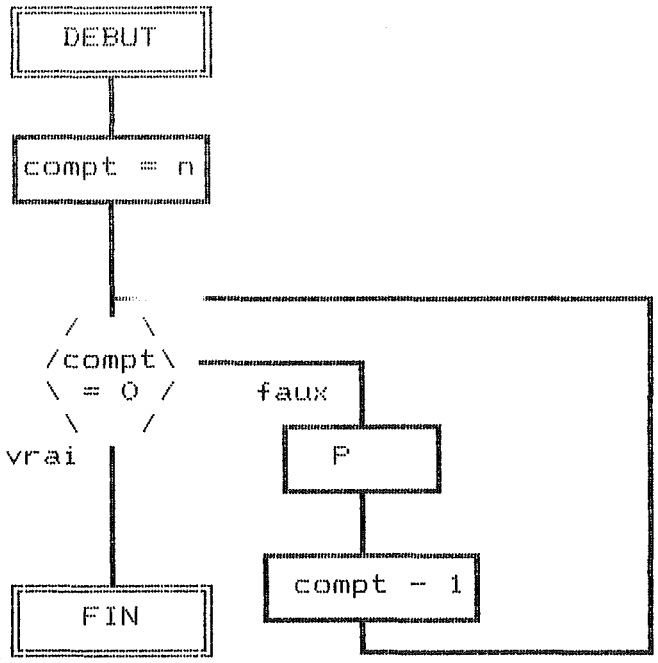
2.2.3.1.3 Le PERFORM option TIMES.

format de départ :

appel-de-procedure ::=

PERFORM proc1
[THRU proc2]
nombre TIMES

organigramme :



Règles de transformation :

- Il faut d'abord effectuer les mêmes actions que pour le perform simple.

- En plus, pour remplacer le TIMES, je vais créer une variable qui sera décrémentée à chaque appel. Le nom de cette variable doit être identifiant et je vais le construire de la manière suivante : soit nbre-time, l'occurrence d'un PERFORM option TIMES dans le programme, la variable à créer sera la concaténation de NUM-TIMES et de nbre-time.
Cette variable sera déclarée en WORKING-STORAGE SECTION.

format résultat :

appel-de-procedure* ::=

MOVE nbre-appel TO VAR|proc
MOVE nombre to NUM-TIMES|nbre-time.
RET|proc|nbre-appel.

```
                    IF ( NUM-TIMES|nbre-time > 0 )
                       SUBSTRACT 1 TO NUM-TIMES|nbre-time
                       GO TO proc1

nouvelles-déclarations ::=
                          nouvelles-déclarations
                          01 VAR|proc  PIC 99.
                          01 COMPT|nbre-time PIC 99.
```

2.2.3.1.4 Le PERFORM option VARYING.

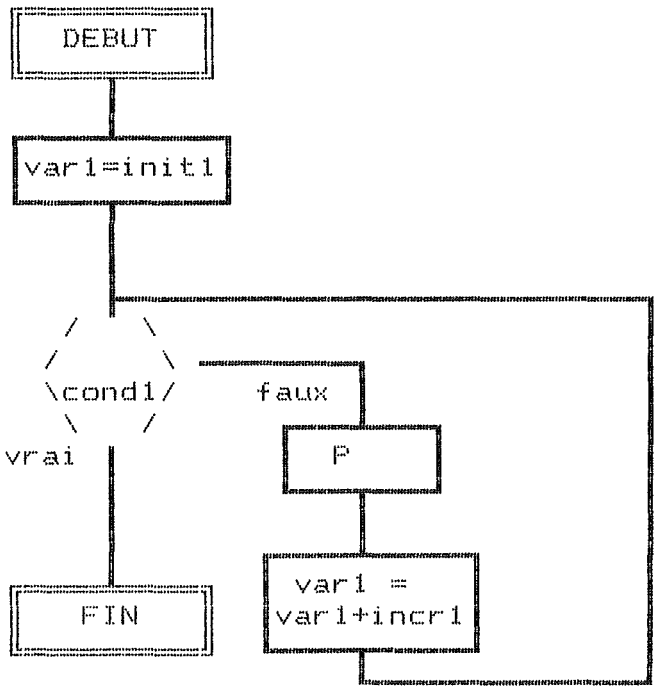
format de départ, 1er cas :

appel-de-procedure ::=

```
        PERFORM proc1
        [ THRU proc2 ]
        VARYING var1 FROM init1 BY incr1 UNTIL (cond1)
```

Rappelons que la condition doit être délimitée par des parenthèses.

organigramme :



Règles de transformation :

- Il faut d'abord effectuer les mêmes actions que pour le perform simple.

- Je vais devoir ensuite construire une 2ème adresse de retour, pour pouvoir incrémenter var1 avant d'effectuer le test mais après le retour normal de la procédure.
soit donc nbre-var, l'occurrence de ce PERFORM VARYING dans le programme, l'adresse de retour sera la concaténation de RETVAR et de nbre-var.

- Je vais enfin créer une 3ème adresse de retour (FINVAR|nbre-appel) simplement pour me permettre de caser une procédure qui ne pourra être appelée que par un GO TO.

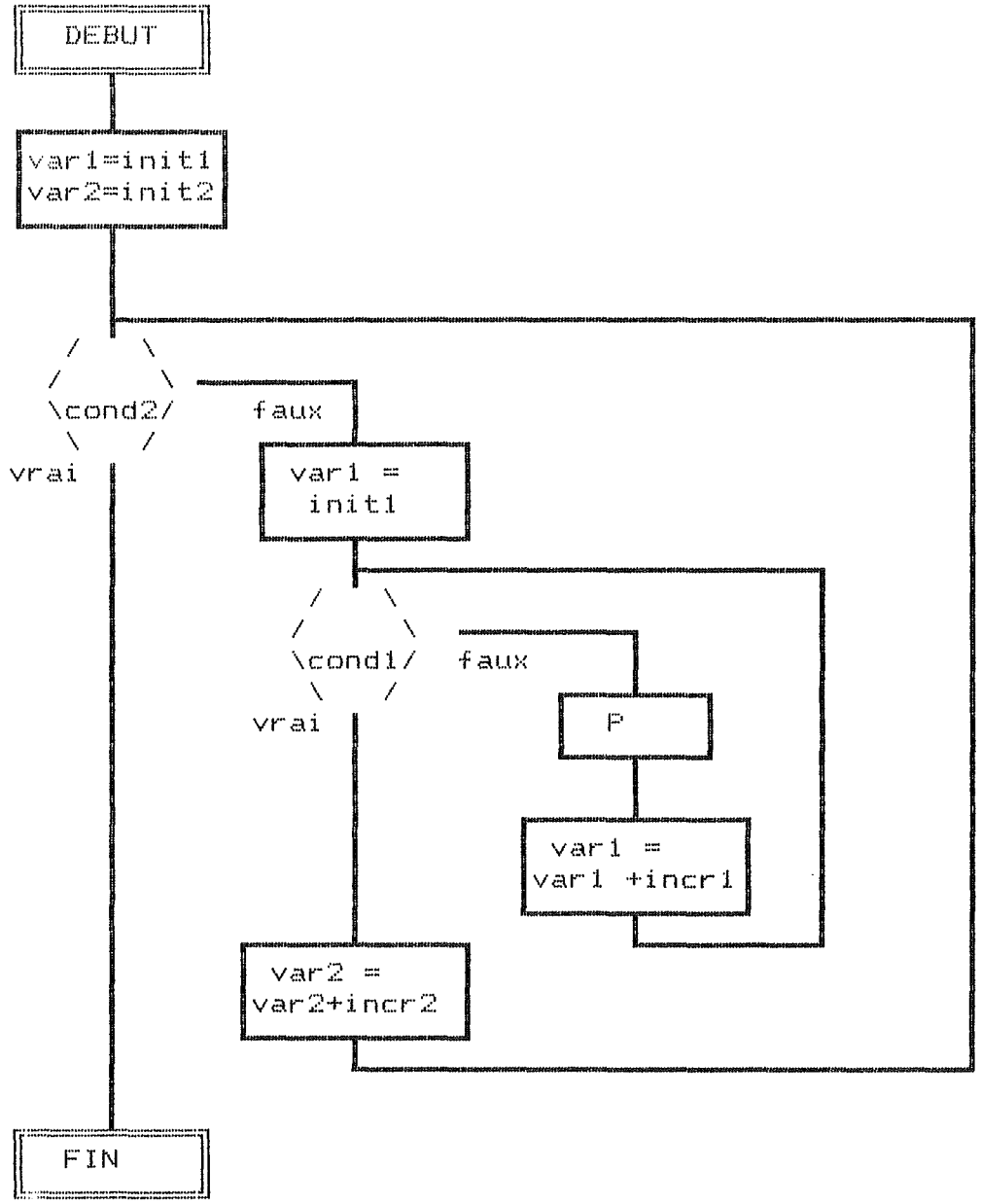
format résultat :

```
appel-de-procedure* ::=
                        MOVE nbre-appel TO VAR|proc
                        MOVE init1 TO var1.

                    RETVAR|nbre-var.
                        IF NOT cond1
                           GO TO proc1.
                        GO TO FINVAR|nbre-appel.

                    RET|proc|nbre-appel.
                        ADD incr1 TO var1
                        GO TO RETVAR|nbre-var.

                    FINVAR|nbre-appel.

nouvelles-déclarations ::=
                         nouvelles-déclarations
                         01 VAR|proc  PIC 99.
```

format de départ, 2eme cas :

```
appel-de-procedure ::=
              PERFORM proc1
              [ THRU proc2 ]
              VARYING var1 FROM init1 BY incr1 UNTIL cond1
              AFTER var2   FROM init2 BY incr2 UNTIL cond2
```

organigramme :



Règles de transformation :

En plus des actions effectuées pour le 1er cas du "PERFORM VARYING", il faudra créer une autre adresse de retour (RET2VAR|nbre-var).

format résultat :

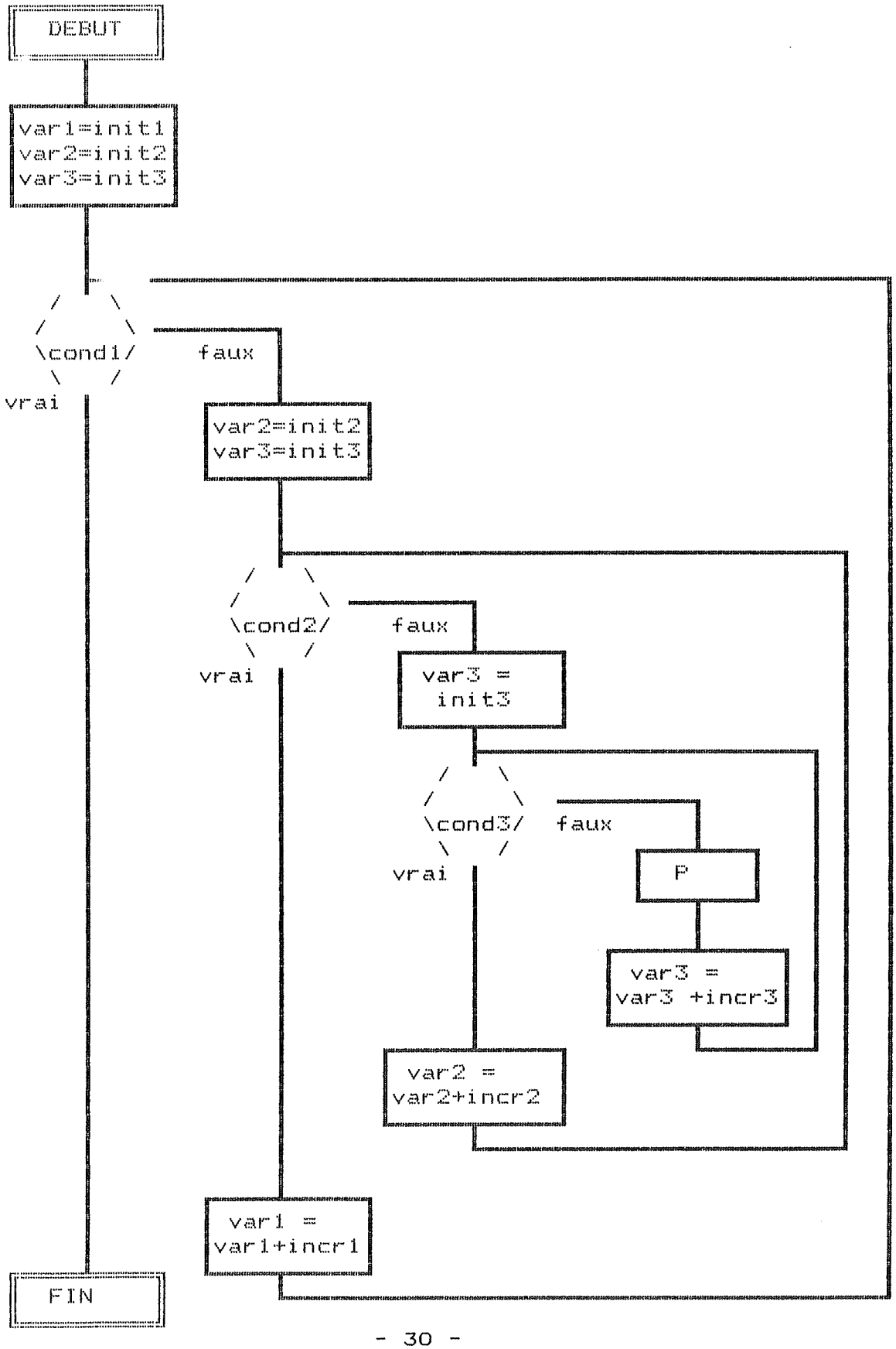
```
appel-de-procedure* ::=
                          MOVE nbre-appel TO VAR|proc
                          MOVE init2 TO var2
                          MOVE init1 TO var1

                       RET2VAR|nbre-var.
                          IF NOT cond2
                             MOVE init1 TO var1
                             GO TO RETVAR|nbre-var.
                          GO TO FINVAR|nbre-appel

                       RETVAR|nbre-var.
                          IF NOT cond1
                             GO TO proc1.
                          ADD incr2 TO var2
                          GO TO RET2VAR|nbre-var

                       RET|proc|nbre-appel.
                          ADD incr1 TO var1
                          GO TO RETVAR|nbre-var

                       FINVAR|nbre-appel.

nouvelles-déclarations ::=
                         nouvelles-déclarations
                         01 VAR|proc  PIC 99.
```

format de départ, 3eme cas :

```
appel-de-procedure ::=
             PERFORM proc1
             [ THRU proc2 ]
```

```
VARYING var1 FROM init1 BY incr1 UNTIL cond1
AFTER var2   FROM init2 BY incr2 UNTIL cond2
AFTER var3   FROM init3 BY incr3 UNTIL cond3
```

organigramme :



Règles de transformation :

En plus des actions effectuées pour le 2eme cas du "PERFORM VARYING", il faudra créer une autre adresse de retour (RET3VAR|nbre-var).

format résultat :

appel-de-procedure* ::=

```
        MOVE nbre-appel TO VAR|proc
        MOVE init3, TO var3
        MOVE init2 TO var2
        MOVE init1 TO var1

    RET3VAR|nbre-var.
        IF NOT cond1
            MOVE init3 TO var3
            MOVE init2 TO var2
            GO TO RET2VAR|nbre-var.
        GO TO FINVAR|nbre-appel

    RET2VAR|nbre-var.
        IF NOT cond2
            MOVE init3 TO var3
            GO TO RETVAR|nbre-var.
        ADD incr1 TO var1
        GO TO RET3VAR|nbre-var

    RETVAR|nbre-var.
        IF NOT cond3
            GO TO proc1.
        ADD incr2 TO var2
        GO TO RET2VAR|nbre-var

    RET|proc|nbre-appel.
        ADD incr3 TO var3
        GO TO RETVAR|nbre-var

    FINVAR|nbre-appel.
```

nouvelles-déclarations ::=

nouvelles-déclarations
01 VAR|proc PIC 99.

2.2.3.2 Le retour.

Le retour s'organisera de la même manière, quel que soit le type de PERFORM.

Le retour consistera, à la fin de chaque procédure et de section, à effectuer un branchement vers l'instruction qui suit le PERFORM appelant la procedure et à ne rien faire si la procedure n'est pas appelée.

Pour organiser le retour, il faut savoir
1) quelles procédures ont été appelées,
2) de combien d'endroits différents du programme elles ont été appelées.

Pour chaque procédure, il faudra effectuer le traitement suivant :
- si la procédure n'a pas été appelée, ne rien changer.

instructions-de-retour sera vide.

- si la procédure a été appelée,
soit nbre-appel, le nombre d'appel à cette procédure,

instructions-de-retour ::=

GO TO RET|proc|1 , RET|proc|2, ...
RET|proc|nbre-appel ON VAR|proc

avec autant d'adresse de retour que de nombre d'appel à cette procédure.

Attention, il ne faut pas considérer les titres de paragraphes créés artificiellement lors du déroutement comme des noms de procédure.

Pour chaque section, le traitement sera le même que pour les procédures, soit :
- si la section n'a pas été appelée, ne rien changer.

instructions-de-retour sera vide.

- si la section a été appelée,
soit nbre-appel, le nombre d'appel à cette section,

instructions-de-retour ::=

GO TO RET|proc|1 , RET|proc|2, ...
RET|proc|nbre-appel ON VAR|proc

avec autant d'adresse de retour que de nombre d'appel à cette section.

2.2.4. La transformation des structures alternatives if.

Comme nous l'avons vu au début du chapitre, nous allons également être obligé de transformer les structures alternatives IF de telle manière que la partie THEN et la partie ELSE ne comprennent que une instruction GO TO.
Nous allons en quelque sorte sortir les instructions qui se trouvent dans la structure IF hors de cette structure.

exemple :

```
IF CONDITION
   INSTRUCT1
ELSE
   INSTRUCT2.
```

sera remplacé par :

```
     IF CONDITION
        GO TO LABEL1
     ELSE
        GO TO LABEL2.
  LABEL1.
     INSTRUCT1*
     GO TO FIN-IF.
  LABEL2.
     INSTRUCT2*
     GO TO FIN-IF.
  FIN-IF.
```

remarque.

INSTRUCT1 ou INSTRUCT2 peut aussi contenir un ou plusieurs IF, et un ou plusieurs PERFORM. Il doit donc aussi être modifié.

format de départ :

```
structure alternative ::=
              IF (condition)
                   instructions1
              [ ELSE instructions2 ] .

instruction1 ::= << instruction >>
instruction2 ::= << instruction >>
```

remarque

Rappelons que la condition doit être délimitée par des parenthèses.

opérations à effectuer :Règles de transformation :

soit nbre-if l'occurrence du IF que l'on transforme dans le programme et qui permettra de créer des labels identifiants.

format résultat :

```
structure-alternative* ::=
                    IF condition
                         GO TO IFTHEN|nbre-if
                  [ ELSE
                         GO TO IFELSE■|nbre-if ].

                IFTHEN|nbre-if.
                       instruction1*
                       GO TO FINIF|nbre-if.

                [ IFELSE|nbre-if.
                       instructions2*
                       GO TO FINIF|nbre-if. ]

                FINIF|nbre-if.


instruction1* ::= << instruction* >>
instruction2* ::= << instruction* >>
```

2.3 L'ANALYSEUR LEXICAL.

Pour faciliter la réalisation des 2 programmes définis aux chapitres précédents, nous allons réaliser un analyseur lexical, qui décomposera le texte cobol en mots.

L'analyseur décomposera donc le texte COBOL en une suite de mots, avec pour chaque mot, un type de mot.
A chaque appel , l'analyseur fournira donc 2 choses :
- une variable de maximum 255 caractères contenant la valeur du mot
- un code du mot sous forme de numéro, représentant le type de mot.

2.3.1 Notation.

La ligne COBOL est découpée en zones. Je parlerai ici de 3 zones :

- **La zone A** est celle où commence le premier mot des titres (en général colonnes 8 à 11)

- **La zone B** contient les phrases constituant le texte du programme. (en général colonnes 12 à 72)

- **La zone C** est occupée par les indicateurs de ligne spéciale (en général la colonne 8).

Les zones situées avant la zone C et après la zone B ne seront pas prises en considération.

2.3.2 Règles de formation des mots.

- si la zone C est occupée par / ou *, il s'agit d'une ligne de commentaire et elle n'est pas prise en considération pour la suite. Elle ne sera pas transmise.

- Si le premier caractère d'un mot se trouve dans la zone A, le type de ce mot sera titre. Ce type sera

décomposé entre plusieurs codes compris entre 10 et 20 selon le mot. exemple : PROCEDURE, WORKING-STORAGE, ...

- Les "séparateurs" ne seront pas transmis comme résultat. Il faudra en tenir compte lors de la reconstruction du texte et donc en insérer entre chaque mot.
Les séparateurs sont :
^ au moins un espace.

^ ou ^,^ virgule suivie d'au moins un espace, éventuellement précédée d'un ou plusieurs espaces

;^ ou ^;^ point-virgule suivit d'au moins un espace, éventuellement précédé d'un ou plusieurs espaces.

De plus, le caractère de fin de ligne sera considéré comme un espace, sauf dans le cas ou la zone C de la ligne suivante contiendra le caractère - (voir coupure et continuation de mots Cobol et littéraux).

- Le point (.) sera par contre transmit .

Si il est suivit d'un espace, il s'agit du point en tant que signe de ponctuation, et il formera seul un mot dont le type sera "point" et le code = 1.

Si il est suivit d'un chiffre, il s'agit de la séparation ente la partie entière et décimale d'un littéral numérique et il sera transmit avec celui-ci.

- les littéraux numériques seront constitués de chiffres, et éventuellement du signe algébrique + ou - en première position, et de la marque décimale (. ou ,). Le code des littéraux numérique sera = 2

- les littéraux non numériques (chaînes de caractères entre guillemets) seront transmis entièrement dans un mot, dont le type sera "chaîne" et le code = 3.
Les espaces contenus dans un littéral non numérique seront bien sur transmis. (voir également coupure et continuation de mots).

- les expressions seront également transmises en une fois, et le type de ce mot sera "expression" et le code = 4.

Une expression sera délimitée par une paire de parenthèses.
Si, à l'intérieur de ces expressions, il y a plusieurs espaces consécutifs, il ne faudra en laisser que 1. Ceci permettra de réduire la longueur de l'expression qui ne pourra pas dépasser la longueur maximum de 255 caractères.

- les opérateurs ou signes monétaire auront le code unique 5. Ce sont les caractères ou suite de caractères suivant :
  + - * / < > = et $.

- les mots suivants recevront chacun un code différent : PERFORM, THRU ou THROUGH (même code), UNTIL, TIMES, VARYING, FROM, BY, AFTER, IF, ELSE, SECTION, OPEN, CLOSE, OUTPUT, READ, WRITE, STOP, RUN, SELECT, FD. (codes > 21)

- les autres mots COBOL recevront un code unique (code = 21).

Un mot Cobol est constitué uniquement de lettres majuscules, de chiffres et de traits d'union, dont ni le premier ni le dernier caractère ne peut être un trait d'union, et contenant en position quelconque au moins une lettre.

Coupure et continuation de mots Cobol et littéraux.

- Lorsque le dernier mot cobol d'une ligne est suivit d'un tiret en zone C de la ligne suivante, cela signifie que ce mot a été coupé et que le suite de ce mot débute au premier caractère non vide de la zone B de cette nouvelle ligne.

- ceci est valable aussi pour un littéral numèrique.

- si un caractère de fin de ligne est rencontré lors de la lecture d'un littéral numérique, les caractères restant jusqu'a la fin de la zone B seront comptés comme espaces;
un tiret doit se trouver dans la zone C de la ligne suivante
un tiret doit, sur cette ligne de continuation, précéder immédiatement la suite du texte.

2.3.3 règles de reconstruction d'un texte Cobol.

Pour reconstruire le texte Cobol à partir d'une suite de mots, les règles suivantes seront d'application :

Entre chaque mot, on insère un espace.

Si on a pas suffisamment de place pour écrire le mot en entier sur la ligne, on passe à la ligne et on l'écrit dans la zone B.
Exception : si on a un littéral non numérique ou une expression, on écrit le début sur la même ligne, puis sur la ligne suivante, on met - dans la zone C et la suite du mot dans la zone B.

Si le code du mot est "titre", on passe à la ligne et on commence en zone A.

Après un point, on va à la ligne. En effet, un titre est toujours suivit d'un point et une ligne contenant un titre ne peut contenir aucun autre texte.

CHAPITRE III : METHODE D'IMPLEMENTATION.

Pour réaliser l'inversion de 2 programmes (P1 et P2), la succession des étapes sera la suivante :

1) exécution de l'analyseur sur P1
2) exécution du programme "suppression PERFORM" sur P1
3) exécution du programme "inversion de P1" sur P1
4) exécution du programme "constructeur" sur P1

Ces 4 étapes seront exécutées par une seule commande.

5) exécution de l'analyseur sur P2
6) exécution du programme "suppression PERFORM" sur P2
7) exécution du programme "inversion de P2" sur P2
8) exécution du programme "constructeur" sur P2.

Ces 4 étapes seront exécutées par une seule commande.

Il restera alors à compiler les 2 programmes et à effectuer un link.

Dans ce chapître, nous allons présenter ces programmes dans l'ordre de leur exécution, c'est-à-dire d'abord l'analyseur (point 3.1), ensuite le programme de suppression des PERFORM (point 3.2) et enfin les programmes d'inversion (point 3.3)

3.1 L'ANALYSEUR LEXICAL.

3.1.1 Hypothèses de départ.

Nous supposerons que :

- le programme en entrée ne contient plus d'erreurs de compilations,
- la longueur maximale d'une expression entre parentèses est de 255 caractères (sachant que plusieurs espaces consécutifs ne comptent que pour un espace).
- la longueur maximale d'un litéral non numèrique (entre guillemets) est de 255 caractères.
- les conditions se trouvent entre parenthèses.

3.1.2 Description.

Ce programme va, à partir d'un fichier de caractères, constituer une suite de mots, avec pour chaque mot, un code. Cette suite de mots sera écrite dans un fichier de sortie.

L'analyseur aura donc en entrée un fichier texte Cobol, et en sortie, un fichier dont chaque enregistrement contiendra un mot et un type de mot.

Avant d'aborder l'algorithme, nous allons citer quelques constantes, variables et primitives importantes.

CONSTANTES :

DEBUT-A = 8 début de la zone A
DEBUT-B = 12 début de la zone B
FIN-B = 72 fin de la zone B
DEBUT-C = 7 début de la zone C

VARIABLES :

CAR : le dernier caractère lu
POS : la position dans la ligne de CAR
MOT-LU : le mot que l'on forme
CODE-MOT: le code de ce mot
CODE-CAR: le code du caractère lu

PRIMITIVES :

LIRE-CAR :

Cette primitive va lire le caractère "utile" suivant, le mettre dans CAR et mettre sa position sur la ligne dans POS et le code de ce caractère dans CODE-CAR.
On entend par caractère "utile" les caractères se trouvant dans les zone A et B et par exception un tiret se trouvant dans la zone C ou un caractère de fin de ligne, même si celui-ci se trouve après la zone B.
Cependant, si la zone C contient / ou *, les caractères de cette ligne ne seront pas transmis.

Si le caractère lu est le caractère de fin de ligne (CR), le caractère transmis sera un espace.

Les codes seront les suivants :

1 : point
2 : espace, CR (fin de ligne) ou caractère de tabulation
3 : les lettres
4 : les chiffres
5 : + plus
6 : - moins ou tiret sauf en zone C
7 : tous les caractères qui ne sont pas repris ci-avant ou ci-après
8 : (parenthèse ouvrante
9 :) parenthèse fermante
10: " guillemets
11: , virgule
12: ; point-virgule
13: tiret en zone C

GARDER-CAR :

Cette primitive va ajouter le caractère CAR à MOT-LU, en concaténant à droite et ajouter 1 a la longueur de MOT-LU.

RETIRER-CAR :

Cette primitive va retirer de MOT-LU le dernier caractère.

ECRIRE-MOT :

Cette primitive va écrire MOT-LU et réinitialiser la longueur de MOT-LU à 0.

3.1.3 Algorithme.

L'algorithme est écrit sous forme de table de décision.

Cette table et les actions à effectuer se trouvent en annexe.
Nous allons ici détailler les étapes de l'analyseur et leur enchaînement.

DESCRIPTION DES ETAPES

Lorsque l'on arrive à une étape, la situation est la suivante : - mot-lu contient la suite de caractère déjà lue

- car-lu contient le caractère suivant.

ETAPE 1 : DEBUT-MOT .

Suite de caractères déjà lue : vide.

Si car-lu est :

- un **point**, on le met dans mot-lu et on va dans l'étape 4 (reçu-point)

- un **espace** ou un point-virgule : il ne faut pas le garder, lire le caractère suivant et rester dans l'étape 1,

- une **virgule**, il faut la mettre dans mot-lu, et passer dans l'étape 5 (reçu-virgule),

- une **lettre**, il faut la mettre dans mot-lu, attribuer le code 10 au mot si la position de ce caractere est dans la zone A, attribuer le code 20 sinon, et aller dans l'étape 2 (mot-cobol),

- un **chiffre**, il faut le mettre dans mot-lu, attribuer le code 10 au mot si la position de ce caractere est dans la zone A et aller dans l'étape 2, sinon aller dans l'étape 3 (littéral-numérique),

- un **signe** (+ ou -), il faut le mettre dans mot-lu et aller dans l'étape 6 (reçu-signe),

- une **parenthèse ouvrante**, il faut la mettre dans mot-lu, initialiser le compteur de parenthèse à 1 et aller dans l'étape 7 (expression),

- des **guillemets**, il faut les mettre dans mot-lu et aller dans l'étape 8 (littéral-non-numérique)

- **autre chose** : il faut le mettre dans mot-lu, code = 5 et le transmettre puis lire le caractère suivant et rester dans l'étape 1.

ETAPE 2: MOT-COBOL .

Situation de départ :

Nous lisons un mot Cobol, dont le code est >= 10, c'est-à-dire un mot formé de lettres, de chiffre et de tirets, sauf en 1ére position.

Ce mot peut être un titre (si code = 10) ou un mot du programme (si code = 20).

Nous quittons cet étape si nous rencontrons un séparateur :

- **virgule, point** ou **point-virgule** : on attribue le code définitif, on écrit le mot et on retourne dans l'étape 1

- **espace** : on lit les caractères suivants jusqu'à rencontrer un caractère >< espace.

  * Si ce caractère a le code 0, cela signifie une continuation de ligne. Il faut a nouveau lire les caractères suivants jusqu'à rencontrer un caractère >< espace puis rester dans l'étape 2.

  * Si ce caractère n'a pas le code 0, il faut attribuer le code définitif au mot-lu, l'écrire et retourner dans l'étape 1.

ETAPE 3: LITTERAL NUMERIQUE .

Situation de départ:

nous lisons un littéral numérique et nous avons déjà lu au moins un chiffre.

Les différents cas possibles sont de recevoir :

- un **chiffre** : il faut le garder, lire le caractère suivant et rester dans l'étape 3,

- un **point** :

-suivi d'un chiffre et il faut le garder et rester dans l'étape 3,

- suivi d'un espace et il faut transmettre le littéral numérique lu, puis le point qui est un séparateur

- une **lettre** ou un **tiret** : le mot-lu est donc un mot-Cobol dont les premiers caractères sont des chiffres; il faut donc passer dans l'étape 2.

- un **espace** : il faut lire les caractères suivants jusqu'à rencontrer un caractère >< espace.

* Si ce caractère a le code 0, cela signifie une continuation de ligne. Il faut a nouveau lire les caractères suivants jusqu'à rencontrer un caractère >< espace puis rester dans l'étape 2.

* Si ce caractère n'a pas le code 0, il faut attribuer le code 2 au mot-lu, l'écrire et retourner dans l'étape 1.

ETAPE 4 : RECU-POINT .

suite de caractères déjà lue : un point.

si le car suivant est :

- un **espace** : ce point est un séparateur et il faut le transmettre, puis retourner dans l'étape 1.

- un **chiffre** : le point marque le début de la partie décimale d'un littéral numérique et on va donc dans l'étape 3.

ETAPE 5: RECU-VIRGULE :

suite de caractères déjà lue : une virgule.

- si le car suivant est :

- un **espace** : cette virgule est un séparateur et on ne le transmet pas. On retourne dans l'étape 1.

- un **chiffre** : la virgule marque le début de la partie décimale d'un littéral numérique et on va donc dans l'étape 3.

ETAPE 6: RECU-SIGNE .

suite de caractères déjà lue : un plus ou un moins.

ce signe reçu en début de mot est

- soit un **opérateur**, et il sera transmis seul, puis lecture du caractère suivant et retour a l'étape 1,

- soit le signe d'un **littéral numérique** si il est suivit d'un chiffre ou d'un point et on va dans l'étape 3.

ETAPE 7: EXPRESSION.

situation de départ :

- suite de caractères déjà lue : une parenthèse suivie plusieurs caractères
- un compteur de parenthèses qui indique le nombre de parenthèses non fermées.

Nous sommes en train de constituer une expression entre parenthèses.

Si le caractère suivant est :

- une **parenthèse ouvrante** : on l'ajoute au mot-lu, on augmente le compteur de parenthèse de 1, on lit le caractère suivant et on reste dans l'étape 7.

- une **parenthèse fermante** : on l'ajoute au mot-lu, on diminue le compteur de parenthèse de 1; si ce compteur est égal à 0, il s'agit de la fin de l'expression et on transmet le mot-lu, on lit le caractère suivant et on retourne dans l'étape 1 ; si le compteur est > 0, on lit le caractère suivant et on reste dans l'étape 7.

- un **espace** : on l'ajoute au mot-lu, et on lit les caractères suivant jusqu'à rencontrer un caractère différent de espace.

- un **autre caractère** : on l'ajoute au mot-lu, on lit le caractère suivant et on reste dans l'étape 7.

ETAPE 8: LITTERAL NON NUMERIQUE.

- suite de caractères déjà lue : des guillemets suivis éventuellement de plusieurs caractères

Nous sommes en train de lire un littéral non numérique. La fin de ce littéral est marquée par des guillemets.

Si le caractère suivant est :

- des **guillemets** : on transmet le mot-lu, on lit le caractère suivant et on retourne dans l'étape 1.

- un caractère dont **code-car** = 0 (caractère de continuation de ligne), il faudra lire les caractères

jusqu'au caractère suivant des guillemets, et rester dans l'étape 8.

- Dans les **autres cas** , il faut ajouter le caractère au mot-lu, lire le caractère suivant et rester dans l'étape 8.

III.2 SUPPRESSION DES PERFORM ET DES IF.

3.2.1. Description.

La transformation des PERFORM s'effectuera en 2 passes :

- la première recevra le texte Colol, traitera le déroutement et constituera une table des procédures appelées. Le résultat de cette première passe sera donc constitué de 2 choses :
* d'une part le texte Cobol, dans lequel les structures IF auront été remplacées et le déroutement des Perform effectué,
* d'autre part une table contenant chaque procédure appelée et le nombre d'appel à cette procédure.
Cette table aura la structure suivante :

```
01 TABLE.
   02 PROC OCCURS 100.
      03 NOM-PROCEDURE  PIC X(30).
      03 NOMBRE-APPEL   PIC 999.
```

* enfin, le nombre d'instructions "PERFORM TIMES" présentes dans le programme. Ce nombre permettra à la seconde passe de déclarer les variables créées lors de la première passe.

- la seconde passe recevra le texte Cobol modifié par la première passe , la table des noms de procédure et le nombre de "PERFORM TIMES".

Sur base de cette table et de ce nombre, elle ajoutera les noms de variables créées lors de la 1ère passe en WORKING-STORAGE SECTION et traitera le retour.
La transformation des structures alternatives IF s'effectuera entièrement en une passe, lors de la première passe de la transformation des PERFORM

3.2.2. Première passe.

3.2.2.1 Algorithme.

tant que la fin du fichier n'est pas atteinte,
lire le mot suivant
si le mot lu est PERFORM, appel à la procédure TRAIT-PERF

si le mot lu est IF, appel à la procédure TRAIT-IF

sinon, écrire le mot lu.

TRAIT-PERF

va lire complètement l'instruction PERFORM,
va ajouter dans la table le nom de la procedure appelée, avec nombre d'appel = 1 si elle ne s'y trouve pas encore, ou ajouter 1 au nombre d'appel si elle s'y trouve déjà,
va remplacer dans le texte Cobol l'instruction PERFORM lue par des instructions équivallentes (voir 2.2.3, le déroutement).

TRAIT-IF

La procédure traitera la structure IF complète, y compris les structure PERFORM et IF éventuelles qui seraient contenues dans cette structure.

3.2.2.2 Description des étapes de la transformation des PERFORM (TRAIT-PERF).

ETAPE 1.

suite de mots reçue : PERFORM
traitement à effectuer : lire PROC, la procédure appelée

Il faudra lire le mot suivant, le mettre dans la variable proc1 (pour le déroutement) et dans la variable proc (pour le retour) et passer à l'étape 2

ETAPE 2.

suite de mots reçue : PERFORM proc1

ou PERFORM proc1 THRU proc2

traitement à effectuer :

- si le mot suivant est THRU :
  il faudra lire le mot suivant et le mettre dans la variable proc (pour le retour) et rester dans l'étape 2

- si le mot suivant est différent de THRU :
  lire dans la table des titres de procédure si PROC s'y trouve ; si oui, ajouter 1 à NBRE-APPEL ; si non, l'y mettre avec NBRE-APPEL = 1.

- si le mot suivant est UNTIL :
  il faudra lire le suivant, le mettre dans la variable cond, générer le texte Cobol correspondant à ce cas (point 2.2.3.1.2) et sortir de la table.

- si le mot suivant est VARYING :
  il faudra lire le (les) mot(s) suivant(s) jusqu'à FROM et les mettre dans la variable var1, puis lire les suivants jusqu'à BY et les mettre dans la variable init1, puis lire les suivants jusqu'à UNTIL et les mettre dans la variable incr1, puis lire le mot suivant (une expression entre (), et le mettre dans cond1
  et enfin aller dans l'étape 3

- si le mot suivant est un littéral numérique :
  il faudra le copier dans la variable nombre et aller dans l'étape 5.
  En effet, il faut déterminer si ce littéral numérique est suivit de TIMES ou pas.

- si le mot suivant est un mot Cobol (code-mot = 20) :
  il faudra le copier dans la variable nombre et aller dans l'étape 6.

- dans les autres cas :

il faudra générer le texte Cobol correspondant à ce cas (point 2.2.3.1.1) et sortir de la table.

ETAPE 3

suite de mots reçue : PERFORM proc1 [THRU proc2]
VARYING var1 FROM init1 BY incr1 UNTIL cond1

traitement à effectuer :

- si le mot suivant est VARYING :
  il faudra lire le (les) mot(s) suivant(s) jusqu'à FROM et les mettre dans la variable var2,
  puis lire les suivants jusqu'à BY et les mettre dans la variable init2,
  puis lire les suivants jusqu'à UNTIL et les mettre dans la variable incr2,
  puis lire le mot suivant (une expression entre (), et le mettre dans cond2
  et enfin aller dans l'étape 4

- dans les autres cas :
  il faudra générer le texte Cobol correspondant à ce cas (point 2.2.3.1.4, 1er cas) et sortir de la table.

ETAPE 4

suite de mots reçue : PERFORM proc1 [THRU proc2]
VARYING var1 FROM init1 BY incr1 UNTIL cond1
AFTER var2 FROM init2 BY incr2 UNTIL cond2

traitement à effectuer :

- si le mot suivant est VARYING :
  il faudra lire le (les) mot(s) suivant(s) jusqu'à FROM et les mettre dans la variable var3,
  puis lire les suivants jusqu'à BY et les mettre dans la variable init3,

puis lire les suivants jusqu'à UNTIL et les mettre dans la variable incr3,
puis lire le mot suivant (une expression entre (), et le mettre dans cond3
ensuite, il faudra générer le texte Cobol correspondant à ce cas (point 2.2.3.1.4, 3ème cas) et sortir de la table.

- dans les autres cas :
il faudra générer le texte Cobol correspondant à ce cas (point 2.2.3.1.4, 2ème cas) et sortir de la table.

ETAPE 5.

suite de mots reçue : PERFORM proc1 [THRU proc2] nombre
avec nombre = littéral numérique.

traitement à effectuer :

- si le mot suivant est TIMES :
il faudra générer le texte Cobol correspondant à ce cas (point 2.2.3.1.3) et sortir de la table.

- sinon (dans ce cas, "nombre" ne fait pas partie de l'instruction PERFORM) :
il faudra générer le texte Cobol correspondant à ce cas (point 2.2.3.1.1), puis écrire la variable nombre et sortir de la table.

ETAPE 6.

suite de mots reçue : PERFORM proc1 [THRU proc2] mot-cobol
avec mot-cobol = un mot dont le code est 20

traitement à effectuer :

- si le mot suivant est TIMES :

il faudra générer le texte Cobol correspondant à ce cas (point 2.2.3.1.3) et sortir de la table.

- si le mot suivant est une expression entre () suivie de TIMES :

  il faudra ajouter cette expression entre () à la variable nombre, générer le texte Cobol correspondant à ce cas (point 2.2.3.1.3) et sortir de la table

- dans les autres cas ("mot-cobol" ne fait pas partie de l'instruction "PERFORM") :

  il faudra générer le texte Cobol correspondant à ce cas (point 2.2.3.1.1), puis écrire "mot-cobol" et sortir de la table.

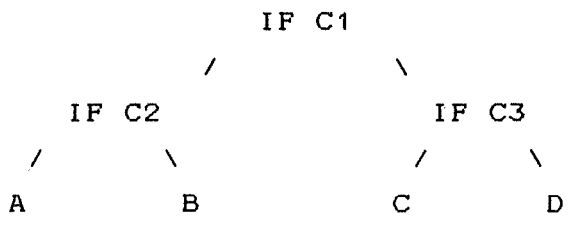
3.2.2.3 Description de la transformation des IF (TRAIT-IF).

3.2.2.3.1 Structure IF.

Une structure IF se termine toujours par un point.
Dns une structure de IF imbriqués, un ELSE se rapporte toujours au dernier IF non encore apparié.

Exemple :

```
IF C1 IF C2 A ELSE B ELSE IF C3 C ELSE  D.
```



Lors du traitement du début d'une structure IF, nous allons créer une pile dans laquelle nous mettrons le numéro de

l'occurence de l'instruction IF rencontrée et une variable qui indiquera que ce IF n'a pas encore été apparié à un ELSE.

```
02 PILE OCCURS 99.
   03 NUM-IF  PIC 99.
   03 APPARIE PIC 9.
```

Ensuite, lors de la rencontre d'un IF, la pile sera augmentée de un élément, lors de la rencontre d'un ELSE, la pile sera diminuée d'un élément et lors de la rencontre d'un point, la pile sera vidée.

3.2.2.3.2 Description des étapes de la transformation des IF.

ETAPE 1.

suite de caractères lue : IF

actions à effectuer :
- incrémenter la variable nombre-if de 1,
- mettre nombre-if dans premier-if.
- initialiser la pile avec nombre-if
- écrire le mot lu (IF)
- lire le mot suivant (la condition) et l'écrire
- générer la 1ère partie du texte Cobol, c'est-à-dire :

```
"    GO TO IFTHENnombre-if
     ELSE
     GO TO IFELSEnombre-if.
IFTHENnombre-if.  "
```

- passer à l'étape 2.

ETAPE 2 (traitement du THEN).

suite de caractères lue : IF condition [instructions]

traitement :

lire le mot suivant
- si le mot suivant est un **point** :

nous sommes à la fin de la structure IF et il faut générer le texte Cobol suivant avant de sortir de la table :

pour chaque IF non apparié à un ELSE (voir dans pile), écrire "IFELSEnombre-if."
écrire " FINIFpremier-if" .

- si le mot suivant est **ELSE** :
il faut générer le texte Cobol suivant

GO TO FINIFpremier-if.
IFELSEnombre-if.

ensuite, retirer le dernier élément de la pile (car IF apparié à un ELSE) et aller à l'étape 3.

- si le mot suivant est **IF**,
il faut
- incrémenter la variable nombre-if de 1,
- ajouter cette variable à la pile
- écrire le mot lu (IF)
- lire le mot suivant (la condition) et l'écrire
- générer le texte Cobol suivant :

GO TO IFTHENnombre-if
ELSE GO TO IFELSEnombre-if.
IFTHENnombre-if.

- rester dans l'étape 2.

- si le mot lu est PERFORM :
appel à la procédure TRAIT-PERF.

- dans les autres cas :
écrire le mot lu et rester dans l'étape 2.

ETAPE 3 (traitement du ELSE).

suite de caractères lue : IF condition [instructions]
ELSE

traitement :

lire le mot suivant

- si le mot suivant est un **point** :

  nous sommes à la fin de la structure IF et il faut générer le texte Cobol suivant avant de sortir de la table :

  pour chaque IF non apparié à un ELSE (voir dans pile), écrire "IFELSEnombre-if."
  écrire " FINIFpremier-if" .

- si le mot suivant est **ELSE** :

  il faut générer le texte Cobol suivant

  GO TO FINIFpremier-if.
  IFELSEnombre-if.

  ensuite, retirer le dernier élément de la pile (car IF apparié à un ELSE) et rester à l'étape 3.

- si le mot suivant est **IF**,

il faut

- incrémenter la variable nombre-if de 1,
- ajouter cette variable à la pile
- écrire le mot lu (IF)
- lire le mot suivant (la condition) et l'écrire
- générer le texte Cobol suivant :

  GO TO IFTHENnombre-if
  ELSE GO TO IFELSEnombre-if.
  IFTHENnombre-if.

- aller à l'étape 2.

- si le mot lu est PERFORM :

  appel à la procédure TRAIT-PERF.

- dans les autres cas :

  écrire le mot lu et rester dans l'étape 3.

3.2.3. Suppression des PERFORM : 2ème passe.

Situation de départ :

Le programme tel que modifié par la première passe.
La table des procédures (TABLE) appelées fournie par la première passe.
Le nombre de "PERFORM TIMES" (nombre-times) fourni par la première passe.

fonction :

va ajouter en WORKING-STORAGE SECTION les déclaration des variables crées lors de la première passe grâce à la table des procédures et à "nombre-times", et organisera le retour, décrit au point 2.2.3.2.

Situation finale :

Un programme Cobol dont l'exécution est semblable à celle du programme Cobol reçu par la première passe.

Algorithme.

- recopier la source jusqu'a WORKING-STORAGE SECTION.

- lire sequentiellement la table des titres et pour chaque élément de cette table, construire le texte suivant :
01 VAR|nom-titre PIC 99.

- lire le nombre de TIMES et faire
POUR i = 1 TO nombre-times
écrire 01 COMPT|i PIC 99.

-recopier la source jusqu'a PROCEDURE DIVISION.

initialiser les variables NOM-PROC et NOM-SECTION a SPACES

tant que l'on est pas à la fin du fichier,
faire
- si le mot lu est un titre dont les 3 premières lettres ne sont pas "WWW", c'est-à-dire un titre qui n'a pas été créé lors de la première passe, faire :

- si la variable NOM-PROC n'est pas vide, écrire le texte du retour et réinitialiser la variable NOM-PROC

- rechercher si le titre se trouve dans la table
si oui, le mettre dans NOM-PROC et mettre le nombre d'appel dans NBRE-APPEL-PROC.

- lire le mot suivant
- si ce mot est "SECTION"
    - si la variable NOM-SECTION n'est pas vide, écrire le texte du retour et réinitialiser la variable NOM-SECTION

    - rechercher si le titre se trouve dans la table
    si oui, le mettre dans NOM-SECTION et mettre le nombre d'appel dans NBRE-APPEL-SECTION.

- écrire l'avant-dernier mot lu (le titre)

- écrire le dernier mot lu

- lire le mot suivant

3.3 L'INVERSION

Les programmes d'inversion seront au nombre de quatre :
Dans le cas de l'inversion de P1 par rapport à P2 (cas 1), nous aurons un programme qui effectuera l'inversion de P1 et un autre qui effectuera les modifications à apporter à P2.
Dans le cas de l'inversion de P2 par rapport à P1 (cas 2), nous aurons un programme qui effectuera l'inversion de P2 et un autre qui effectuera les modifications à apporter à P1.

3.3.1 Inversion de P1 (cas 1).

Les modifications à apporter ont été décrites au point 2.1.2.2.
Cette inversion s'effectuera en 2 passes.

Première passe

Entrée :
Le programme P1, après la suppression des PERFORM et des IF.

Fonction :
- Effectuera toutes les modifications décrites au point 2.1.2.2 sauf le remplacement de "PROCEDURE DIVISION." par
"PROCEDURE DIVISION USING EOF REC.
    GO TO Q1 Q2 ... QM+1 DEPENDING ON QS.
 Q1. "
- Comptera en commençant à 2, le nombre d'instructions WRITE REC et mettra ce nombre dans NBRE-WRITE.

Résultat :
Le programme P1 modifié
La variable NBRE-WRITE contenant le nombre d'instructions "WRITE REC" de P1 + 1.

Seconde passe

Entrée :

le programme Cobol modifié par la première passe

La variable NBRE-WRITE, fournie par la 1ere passe.

Fonction :

Procèdera au remplacement de "PROCEDURE DIVISION." par "PROCEDURE DIVISION USING EOF REC.

GO TO Q1 Q2 ... QM+1 DEPENDING ON QS.

Q1. "

avec M+1 = NBRE-WRITE.

Résultat :

Le programme P1 inversé. Ce programme devra encore être fourni au constructeur avant d'être compilable.

3.3.2 Modifications de P2 (cas 1).

Ces modifications (décrites au point 2.1.2.1) seront effectuées en une seule passe.

Entrée :

Le programme P2, après la suppression des PERFORM et des IF.

Fonction :

Effectuera toutes les modifications décrites au point 2.1.2.1.

Résultat :

Le programme P2 inversé. Ce programme devra encore être fourni au constructeur avant d'être compilable.

3.3.3 Modifications de P1 (cas 2).

Ces modifications (décrites au point 2.1.3.1) seront effectuées en une seule passe.

Entrée :

Le programme P1, après la suppression des PERFORM et des IF.

Fonction :

Effectuera toutes les modifications décrites au point 2.1.2.2.

Résultat :

Le programme P1 inversé. Ce programme devra encore être fourni au constructeur avant d'être compilable.

3.3.4 Inversion de P2 (cas 2).

Les modifications à apporter ont été décrites au point 2.1.3.2.
Cette inversion s'effectuera en 2 passes.

Première passe

Entrée :

Le programme P2, après la suppression des PERFORM et des IF.

Fonction :

- Effectuera toutes les modifications décrites au point 2.1.3.2 sauf le remplacement de "PROCEDURE DIVISION." par
"PROCEDURE DIVISION USING EOF REC.
GO TO Q1 Q2 ... QM+1 DEPENDING ON QS.
Q1. "
- Comptera en commençant à 2, le nombre d'instructions READ FICHC et mettra ce nombre dans NBRE-READ.

Résultat :

Le programme P2 modifié

La variable NBRE-READ contenant le nombre d'instructions "READ FICHC" de P2 + 1.

Seconde passe

Entrée :

le programme Cobol modifié par la première passe

La variable NBRE-READ, fournie par la 1ere passe.

Fonction :

Procèdera au remplacement de "PROCEDURE DIVISION." par "PROCEDURE DIVISION USING EOF REC.

```
    GO TO Q1 Q2 ... QM+1 DEPENDING ON QS.
 Q1. "
```

avec M+1 = NBRE-READ.

Résultat :

Le programme P2 inversé. Ce programme devra encore être fourni au constructeur avant d'être compilable.

CHAPITRE IV. GESTIONNAIRE D'UNE BIBLIOTHEQUE DE PROGRAMMES PERMETTANT L'INVERSION EN CHAINE.

4.1 DEFINITION DE LA BIBLIOTHEQUE.

Ce gestionnaire permettra de consulter et de mettre à jour une liste de programmes disponibles.
Cette liste comprendra le nom du programme, le nom interne de ses fichier séquentiels d'entrée/sortie ainsi qu'une brève description de sa constitution si il est lui-même le résultat d'une inversion.

exemple :

```
    nom                 : A
    fichier(s) entrée   : FICH1
    fichier(s) sortie   : FICH2

    nom                 : B
    fichier(s) entrée   : FICH3
    fichier(s) sortie   : FICH4
```

Nous pouvons aussi envisager d'ajouter une ligne de commentaire qui décrira brièvement le programme.
exemple :

```
    nom                 : A  = analyseur lexiqual
                               d'un texte Cobol.
    fichier(s) entrée   : FICH1
    fichier(s) sortie   : FICH2
```

Ce gestionnaire permettra aussi de procéder à une inversion de plusieurs programmes de la liste et d'ajouter automatiquement le résultat de cette inversion dans la liste.

exemple d'inversion de 2 programmes :

programmes à inverser	fichier
A	FICH1
B	FICH3

résultat : C

ce qui signifie :
inverser A par rapport a sa sortie FICH2
et B par rapport à son entrée FICH3, et mettre le résultat dans C.
Le gestionnaire a le choix du sens de l'inversion et après l'inversion, le gestionnaire ajoutera à la liste :

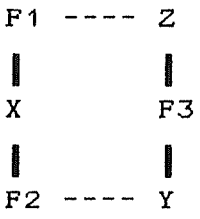
nom : C = A^{-1} FICH2-FICH3 B
fichier(s) entrée :
fichier(s) sortie : FICH4

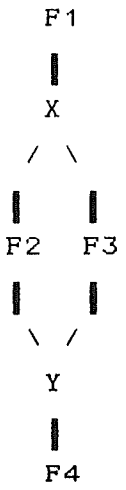
A-1 signifie que A est devenu un sous-programme.

4.2 INVERSION EN CHAINE.

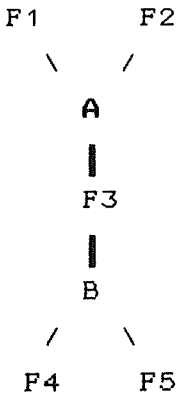
4.2.1 Conditions.

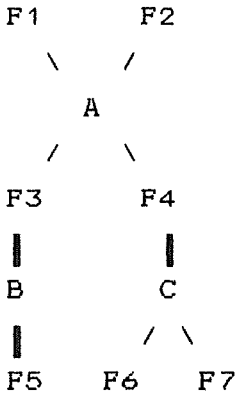
Il sera possible de réaealiser des inversion en chaine.
Les 2 conditions pour que une inversion soit possibles sont :
1) les programmes à inverser doivent être séparés par un et un seul fichier intermédiaire séquentiel,
2) il ne peut pas y avoir une boucle dans le schéma de l'inversion
exemples non valide :





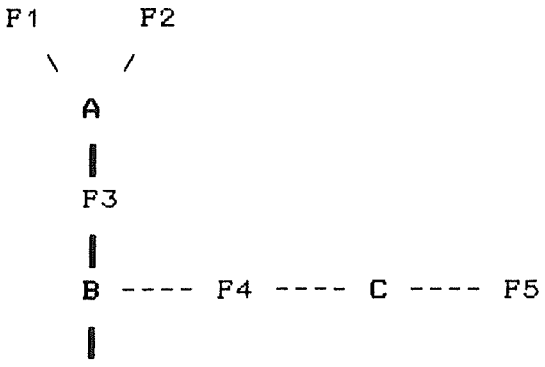
exemples valides

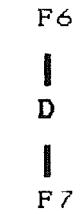




4.2.2 Exemple.

soit la structure de programmes suivante :





La description de ce schéma dans la bibliothèque est la suivante:

```
nom                : A
fichier(s) entrée  : F1 , F2
fichier(s) sortie  : F3

nom                : B
fichier(s) entrée  : F3
fichier(s) sortie  : F4 , F6

nom                : C
fichier(s) entrée  : F4
fichier(s) sortie  : F5

nom                : D
fichier(s) entrée  : F6
fichier(s) sortie  : F7
```

Si nous voulons procéder à l'inversion complète , il faudra donner au gestionnaire la liste des inversions à effectuer et le nom du programme résultat :

```
A  F3 - F3  B
B  F4 - F4  C
B  F6 - F6  D
```

résultat dans E.

Si les inversions sont possibles, c'est-à-dire si il n'y a pas de boucle dans le shema, le gestionnaire procèdera à l'inversion totale et ajoutera résultat dans la bibliothèque

```
nom                : E = A-1 F3-F3 B F4-F4 C-1,
```

B F6-F6 D-1

fichier(s) entrée :

fichier(s) sortie :

4.2.3 Méthode.

Comment va-t-il procéder pour trouver l'ordre et le sens des inversions?

Nous allons partir de la règle suivante :
Un programme ne peut pas être sous-programme de 2 programmes différents.
Sachant cela, nous allons commencer par inverser les programmes ne se trouvant que une fois dans la liste, en transformant ces programmes en sous-programmes.
il y en a au moins un car il ne peut y avoir de boucle.
Ces programmes deviendront des sous-programmes et la liste sera donc réduite. Il suffira de recommencer la même opération jusqu'a ce qui la liste ne contienne plus que un seul élément.
Dans l'exemple développé ici, les inversions successives seront les suivantes,

A-1 B
B C-1
B D-1

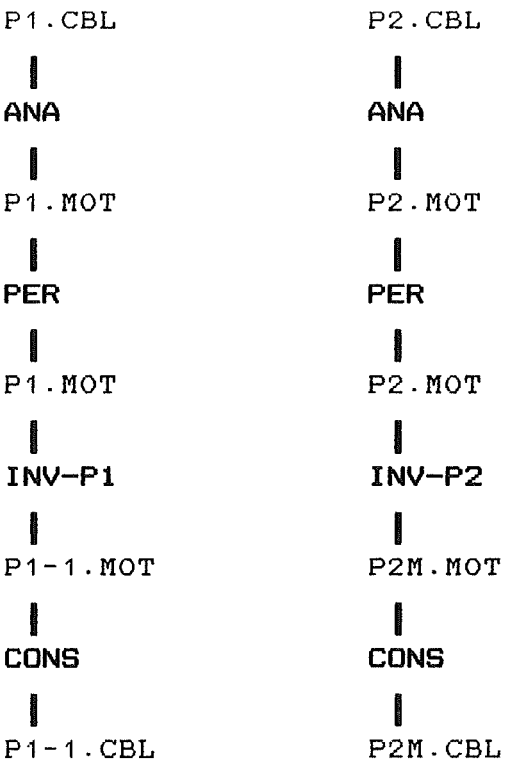
et le résultat sera :

E = ((A-1 B) C-1) D-1

AUTRE EXEMPLE.

Dans le chapitre précédent, nous avons réalisé un analyseur (ana), un programme de transformation des perform et if (per), un programme d'inversion automatique pour P1 et pour P2 (inv-P1 et inv-P2) et enfin un programme de reconstruction du texte COBOL (cons).
Ces programmes seront exécutés successivement et sont séparés entre eux par des fichiers séquentiels.
Il sera dès lors possible de les inverser.

Le schéma de la structure d'enchainement de ces programmes, dans le cas de l'inversion de P1 par rapport à P2, est le suivant :



Si nous inversons le tout, le gain sera considérable : Tous les fichiers intermédiaires entre ces programmes disparaitront.
Les seuls fichiers intermédiaires qui seront maintenus seront ceux entre les 2 passes du programme de transformation des PERFORM (per) et ceux entre les 2 passes du programme d'inversion de P1 (inv-P1). En effet, les 2 passes ne peuvent évidemment pas être inversées car elles sont reliées par un fichier séquentiel mais aussi une table sous forme d'un fichier indexé.

4.2.4 Algorithme de l'inversion en chaîne.

situation de départ :

- une table qui contient les paires de programmes à inverser et les noms des fichiers intermédiaires.

```
01 TABLE-INVERSIONS.
   02 NOMBRE-ELEMENT PIC 99.
```

```
02 ELEMENT-TABLE OCCURS 99.
   03 P1      PIC X(9).
   03 FP1     PIC X(9).
   03 P2      PIC X(9).
   03 FP2     PIC X(9).
```

Traitement :

tant qu'il y a des éléments dans la table,

à partir de cette table, créer une liste des noms de programmes, avec le nombre d'apparition de ces programmes dans la table (c'est-à-dire le nombre de fois qu'ils doivent être inversés).

pour chaque élément de la liste dont le nombre d'inversion = 1,

rechercher ce programme dans la table
procéder à l'inversion en faisant de ce programme un sous-programme,
le supprimer de la table et de la liste.

4.2.5.

Une étape suivante consisterait à réaliser l'intégration de l'inversion en chaîne, de la compilation des différents programmes et sous-programmes et du linkage de ces parties pour obtenir finalement un programme exécutable.

ANNEXE 1. Exemple de programmes

1. programme P1
2. programme P2
3. programme P1 : PERFORM supprimés
4. programme P2 : PERFORM supprimés
5. programme P1-1 : P1 inversé
6. programme P2M : P2 modifié

```
 IDENTIFICATION DIVISION.
*************************
*
*   but de ce programme : lire les elements du fichier bande
*                         et les ecrires dans fich-c séparés
*                         par des *.
*
*
 PROGRAM-ID. P1.

 ENVIRONMENT DIVISION.
***********************
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

 SELECT FICH-C ASSIGN TO "INTER.CBL"
 ORGANIZATION IS SEQUENTIAL.

 SELECT BANDE ASSIGN TO "BANDE.CBL"
 ORGANIZATION IS SEQUENTIAL.

 DATA DIVISION.
****************
 FILE SECTION.

 FD FICH-C LABEL RECORD IS STANDARD.
 01 REC   PIC X.

 FD BANDE LABEL RECORD IS STANDARD.
 01 COCO   PIC X.

 WORKING-STORAGE SECTION.

 01 A PIC 999.
 01 FIN PIC 9.

 PROCEDURE DIVISION.
*********************
     OPEN OUTPUT FICH-C.
     OPEN INPUT BANDE.

     MOVE 0 TO FIN.
     PERFORM TRAIT-BANDE UNTIL (FIN = 1).
     CLOSE FICH-C
     CLOSE BANDE
     STOP RUN.

 TRAIT-BANDE.
*************
     READ BANDE  RECORD AT END MOVE 1 TO FIN.
     IF (FIN = 0)
        WRITE REC  FROM COCO
        MOVE "-" TO REC
        WRITE REC.
```

```
 IDENTIFICATION DIVISION.
************************
*
*   but de ce programme P2 : lire les éléments du fichier fich-c
*   et les écrires dans rapport séparés par un tiret.
*

 PROGRAM-ID. TP2.

 ENVIRONMENT DIVISION.
**********************
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

 SELECT FICH-C ASSIGN TO "INTER.CBL"
 ORGANIZATION IS SEQUENTIAL.

 SELECT RAPPORT ASSIGN TO "RAPPORT.CBL"
 ORGANIZATION IS SEQUENTIAL.

 DATA DIVISION.
***************
 FILE SECTION.

 FD FICH-C LABEL RECORD IS STANDARD.
 01 REC   PIC X.

 FD RAPPORT LABEL RECORD IS STANDARD.
 01 COCO   PIC X.

 WORKING-STORAGE SECTION.

 01 A PIC 999.
 01 FIN PIC 9.

 PROCEDURE DIVISION.

      OPEN INPUT FICH-C.
      OPEN OUTPUT RAPPORT.
      MOVE 0 TO FIN.
      PERFORM TRAIT-FICH UNTIL (FIN = 1).
      CLOSE FICH-C
      CLOSE RAPPORT
      STOP RUN.

 TRAIT-FICH.
*************
      READ FICH-C RECORD AT END MOVE 1 TO FIN.
      IF (FIN = 0)
         WRITE COCO FROM REC
         MOVE "*" TO COCO
         WRITE COCO.


*
*   ces programme P1 et P2 sont simplistes, mais permettent de
*   montrer le mécanisme de transformation des PERFORM et des
*   IF, ainsi que le mécanisme de l'inversion.
*   ici, ce sera une inversion de P1 par rapport à P2.
*
```

Voici le programme P1 après avoir été analysé et décomposé en mots, après la transformation des PERFORM et des IF, et après avoir été reconstruit par le constructeur.

```
IDENTIFICATION DIVISION .

PROGRAM-ID .
    P1 .

ENVIRONMENT DIVISION .

INPUT-OUTPUT SECTION .

FILE-CONTROL .
    SELECT FICH-C ASSIGN TO "INTER.CBL"
ORGANIZATION IS SEQUENTIAL .
    SELECT BANDE ASSIGN TO "BANDE.CBL"
ORGANIZATION IS SEQUENTIAL .

DATA DIVISION .

FILE SECTION .
    FD
FICH-C LABEL RECORD IS STANDARD .

01
REC PIC X .
    FD
BANDE LABEL RECORD IS STANDARD .

01
COCO PIC X .

WORKING-STORAGE SECTION .

01 VARTRAIT-BANDE                    PIC 99 .

01
A PIC 999.
01
FIN PIC 9.
PROCEDURE DIVISION .
    OPEN OUTPUT FICH-C .
    OPEN INPUT BANDE .
    MOVE 0 TO FIN .
    MOVE 01 TO VARTRAIT-BANDE .

WWWRETTRAIT-BANDE01 .
    IF NOT (FIN = 1) GO TO TRAIT-BANDE .
    CLOSE FICH-C CLOSE BANDE STOP RUN .

TRAIT-BANDE .
    READ BANDE RECORD AT END MOVE 1 TO FIN .
    IF (FIN = 0) GO TO WWWIFTHEN01 ELSE GO TO WWWIFELSE01 .

WWWIFTHEN01 .
    WRITE REC FROM COCO MOVE "-" TO REC WRITE REC .

WWWIFELSE01 .
```

```
WWWFINIFO1 .
    GO TO    WWWRETTRAIT-BANDEO1
     WWWRETTRAIT-BANDEO1
     DEPENDING ON VARTRAIT-BANDE                    .
```

Voici le programme P2 après avoir été analysé et décomposé en mots, après la transformation des PERFORM et des IF, et après avoir été reconstruit par le constructeur.

```
IDENTIFICATION DIVISION .

PROGRAM-ID .
    TP2 .

ENVIRONMENT DIVISION .

INPUT-OUTPUT SECTION .

FILE-CONTROL .
    SELECT FICH-C ASSIGN TO "INTER.CBL"
ORGANIZATION IS SEQUENTIAL .
    SELECT RAPPORT ASSIGN TO "RAPPORT.CBL"
ORGANIZATION IS SEQUENTIAL .

DATA DIVISION .

FILE SECTION .
    FD
FICH-C LABEL RECORD IS STANDARD .

01
REC PIC X .
    FD
RAPPORT LABEL RECORD IS STANDARD .

01
COCO PIC X .

WORKING-STORAGE SECTION .

01 VARNOMBRE-TIME                    PIC 99 .

01 VARTRAIT-FICH                     PIC 99 .

01
A PIC 999.
01
FIN PIC 9.
PROCEDURE DIVISION .
    OPEN INPUT FICH-C .
    OPEN OUTPUT RAPPORT .
    MOVE 0 TO FIN .
    MOVE 01 TO VARTRAIT-FICH .

WWWRETTRAIT-FICH01 .
    IF NOT (FIN = 1) GO TO TRAIT-FICH .
    CLOSE FICH-C CLOSE RAPPORT STOP RUN .

TRAIT-FICH .
    READ FICH-C RECORD AT END MOVE 1 TO FIN .
    IF (FIN = 0) GO TO WWWIFTHEN01 ELSE GO TO WWWIFELSE01 .

WWWIFTHEN01 .
    WRITE COCO FROM REC MOVE "*" TO COCO WRITE COCO .
```

```
WWWIFELSE01 .

WWWFINIF01 .
    GO TO   WWWRETTRAIT-FICH01
     WWWRETTRAIT-FICH01
     DEPENDING ON VARTRAIT-FICH                    .
```

Voici le programme P1 après inversion : il est devenu P1-1.

```
IDENTIFICATION DIVISION .

PROGRAM-ID .
    TP1-1       .

ENVIRONMENT DIVISION .

INPUT-OUTPUT SECTION .

FILE-CONTROL .
    SELECT BANDE ASSIGN TO "BANDE.CBL"
ORGANIZATION IS SEQUENTIAL .

DATA DIVISION .

FILE SECTION .
    FD
BANDE LABEL RECORD IS STANDARD .

01
COCO PIC X .

WORKING-STORAGE SECTION .

01 QS PIC 99 VALUE 1  .

01 VARNOMBRE-TIME                    PIC 99 .

01 VARTRAIT-BANDE                    PIC 99 .

01
A PIC 999.
01
FIN PIC 9.
LINKAGE SECTION .

01
REC PIC X .
    01 EOF PIC 9  .

PROCEDURE DIVISION USING EOF REC                                    .
    GO TO QO1  QO2  QO3  DEPENDING ON QS .

QO1 .
    OPEN INPUT BANDE .
    MOVE O TO FIN .
    MOVE 01 TO VARTRAIT-BANDE .

WWWRETTRAIT-BANDE01 .
    IF NOT (FIN = 1) GO TO TRAIT-BANDE .
    CLOSE BANDE MOVE 1 TO EOF .

DERNIRESORTIE .
    EXIT PROGRAM .

FINPROGRAM .

TRAIT-BANDE .
    READ BANDE RECORD AT END MOVE 1 TO FIN .
```

```
    IF (FIN = 0) GO TO WWWIFTHENO1 ELSE GO TO WWWIFELSEO1 .

WWWIFTHENO1 .
    MOVE 02 TO QS MOVE 0 TO EOF MOVE
    COCO                                TO
    REC                                 .

SORTIEO2  .
    EXIT PROGRAM .

QO2  .
    MOVE "-" TO REC MOVE 03 TO QS MOVE 0 TO EOF .

SORTIEO3  .
    EXIT PROGRAM .

QO3  .

WWWIFELSEO1 .

WWWFINIFO1 .
    GO TO   WWWRETTRAIT-BANDEO1
     WWWRETTRAIT-BANDEO1
     DEPENDING ON VARTRAIT-BANDE                    .
```

Voici le programme P2 après Modification Le nom a changé et est devenu TPR

```
IDENTIFICATION DIVISION .

PROGRAM-ID .
    TPR          .

ENVIRONMENT DIVISION .

INPUT-OUTPUT SECTION .

FILE-CONTROL .
    SELECT RAPPORT ASSIGN TO "RAPPORT.CBL"
ORGANIZATION IS SEQUENTIAL .

DATA DIVISION .

FILE SECTION .
    FD
RAPPORT LABEL RECORD IS STANDARD .

01
COCO PIC X .

WORKING-STORAGE SECTION .

01
REC PIC X .
    01 EOF PIC 9  .

01 VARNOMBRE-TIME                    PIC 99 .

01 VARTRAIT-FICH                     PIC 99 .

01
A PIC 999.
01
FIN PIC 9.
PROCEDURE DIVISION .
    OPEN OUTPUT RAPPORT .
    MOVE 0 TO FIN .
    MOVE 01 TO VARTRAIT-FICH .

WWWRETTRAIT-FICH01 .
    IF NOT (FIN = 1) GO TO TRAIT-FICH .
    IF (EOF = 0)  CALL "TP1P-1C"      USING EOF
    REC                              .
    CLOSE RAPPORT STOP RUN .

TRAIT-FICH .
    CALL "TP1P-1C" USING EOF   REC
    .
    IF (EOF = 1)  MOVE 1 TO FIN .
    IF (FIN = 0) GO TO WWWIFTHEN01 ELSE GO TO WWWIFELSE01 .

WWWIFTHEN01 .
    WRITE COCO FROM REC MOVE "*" TO COCO WRITE COCO .

WWWIFELSE01 .

WWWFINIF01 .
```

```
GO TO    WWWRETTRAIT-FICHO1
 WWWRETTRAIT-FICHO1
 DEPENDING ON VARTRAIT-FICH                                    .
```

ANNEXE 2. Liste des programmes

1. Analyseur (table de décision et programme)
2. Constructeur
3. Suppression des PERFORM, 1ère et 2ème passe
4. Inversion de P1 par rapport à P2,
   Inversion de P1
5. Inversion de P1 par rapport à P2,
   Modification de P2

CODE-CAR / ETAT-PRESENT	1 POINT	2 ESPACE TAB CR	3 LETTRE	4 CHIFFRE	5 +	6 -	7 AUTRE	8 (	9)	10 "	11 ,	12 ;	13 - EN ZONE C
1 DEBUT-MOT	A1 E4	A4 E1	A12 E2	A13	A1 E6	A1 E6	A21 E1	A6 E7		A1 E8	A1 E5	A4 E1	
2 MOT-COBOL	A2 E1	A18	A1 E2	A1 E2		A1 E2		A20	A1		A2 E1	A2 E1	A16 E2
3 LITTERAL-NUM	A11	A14	A1 E2	A1 E3		A1 E2		A20 E2			A5	A14	A16 E3
4 RECU-POINT		A3 E1		A1 E3									A16 E4
5 RECU-VIRGULE		A15 E1		A1 E4									A16 E5
6 RECU-SIGNE	A1 E3	A10 E1		A1 E3							A1 E3		A16 E6
7 EXPRESSION	A1 E7	A17 E7	A1 E7	A1 E7	A1 E7	A1 E7	A1 E7	A7 E7	A8	A1 E7	A1 E7	A1 E7	A16 E7
8 LITTERAL-NON-NUMERIQUE	A1 E8	A22 E8	A1 E8	A1 E8	A1 E8	A1 E8	A1 E8	A1 E8	A1 E8	A9	A1 E8	A1 E8	A19 E8

```
*
*    ANALYSEUR LEXICAL : TRANSFORME UN TEXTE COBOL
*    EN UNE SUITE DE MOT, AVEC POUR CHAQUE MOT,
*    UN TYPE DE MOT.
*
*
*

 IDENTIFICATION DIVISION.
**************************
 PROGRAM-ID.  ANALYSEUR.

*
*
 ENVIRONMENT DIVISION.
***********************
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 SELECT FICHIER-ENTREE  ASSIGN TO FICHIER-SOURCE
        ORGANIZATION IS SEQUENTIAL.

 SELECT FICHIER-SORTIE ASSIGN TO FICHIER-RESULTAT
        ORGANIZATION IS SEQUENTIAL.

*
*
 DATA DIVISION.
****************

 FILE SECTION.
***************
 FD FICHIER-ENTREE LABEL RECORD IS STANDARD.
 01 CAR-FICHIER   PIC X.

 FD FICHIER-SORTIE LABEL RECORD IS STANDARD.
 01 REC-SORTIE.
    02 CODE-SORTIE   PIC 99.
    02 LONGUEUR      PIC 999.
    02 MOT-SORTIE .
       03 CAR-MOT PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONGUEUR.

 WORKING-STORAGE SECTION.
**************************

 01 DEBUT-A    PIC 99 VALUE 8.
 01 DEBUT-B    PIC 99 VALUE 12.
 01 FIN-B      PIC 99 VALUE 72.
 01 DEBUT-C    PIC 99 VALUE 7.

 01 FIN-LI     PIC 99 COMP VALUE 13.
 01 CAR-FIN-LIGNE REDEFINES FIN-LI PIC X.

 01 FICHIER-SOURCE    PIC X(12).
 01 FICHIER-RESULTAT  PIC X(12).

 01 CAR        PIC X.
 01 POS        PIC 99.

 01 CODE-MOT   PIC 99.
 01 CODE-CAR   PIC 99.

 01 COMPT-PARENTHESE   PIC 99.
 01 NBRE-ESPACE        PIC 99.
 01 FIN-LIGNE          PIC 9.
```

```
 01 ETAT-PRESENT    PIC 99.

 01 FIN-FICHIER     PIC 9.

 01 COMPT  PIC 9(5).

*
*
 PROCEDURE DIVISION.
**********************
*
*   Saisie des paramètre FICHIER-SOURCE et FICHIER-RESULTAT
*
*
      DISPLAY "entrez le fichier source (.CBL)" UPON CONSOLE.
      ACCEPT FICHIER-SOURCE FROM CONSOLE.
      MOVE FICHIER-SOURCE TO FICHIER-RESULTAT.
      INSPECT FICHIER-SOURCE REPLACING FIRST "     " BY ".CBL".
      INSPECT FICHIER-RESULTAT REPLACING FIRST "     " BY ".MOT".

      DISPLAY "le fichier source est " FICHIER-SOURCE
              "et le fichier resultat est " FICHIER-RESULTAT
                                              UPON CONSOLE.

*
*   Début du programme proprement dit
*
*
      OPEN INPUT FICHIER-ENTREE.
      OPEN OUTPUT FICHIER-SORTIE.

      MOVE SPACES TO MOT-SORTIE
      MOVE 0 TO LONGUEUR
      MOVE 0 TO POS
      PERFORM LIRE-CAR
      MOVE 1 TO ETAT-PRESENT
      MOVE 0 TO FIN-FICHIER
         MOVE 0 TO FIN-LIGNE

      MOVE 0 TO COMPT

      GO TO DEBUT-TABLE.

 RETOUR-TABLE.

      CLOSE FICHIER-ENTREE.
      CLOSE FICHIER-SORTIE.
      STOP RUN.

*                                                              *
*          P R I M I T I V E S                                 *
*                                                              *

 LIRE-CAR.
**********
*
*   cette primitive va lire le caractère utile suivant, le mettre
*   dans CAR, mettre sa position sur la ligne dans POS et le code
*   de ce caractère dans CODE-CAR;
*   La notion de caractère utile et les différents code sont
*   expliqués dans le chapître III, point 3, PRIMITIVES
*
      MOVE 0 TO CODE-CAR.
      PERFORM LIRE-SUIVANT.
```

```
      IF (FIN-FICHIER = 0)
         PERFORM TRAIT-DEBUT-LIGNE
         UNTIL (( POS NOT < DEBUT-A ) AND ( POS NOT > FIN-B )).

      IF CODE-CAR  = 13
            NEXT SENTENCE
      ELSE IF CAR = "."
            MOVE 1 TO CODE-CAR
      ELSE IF CAR = " " OR CAR = CAR-FIN-LIGNE OR CAR = "  "
            MOVE 2 TO CODE-CAR
      ELSE IF (CAR NOT < "A" AND CAR NOT > "Z") OR
              (CAR NOT < "a" AND CAR NOT > "z")
            MOVE 3 TO CODE-CAR
      ELSE IF CAR NOT < "0" AND CAR NOT > "9"
            MOVE 4 TO CODE-CAR
      ELSE IF CAR = "+"
            MOVE 5 TO CODE-CAR
      ELSE IF CAR = "-"
            MOVE 6 TO CODE-CAR
      ELSE IF CAR = "("
            MOVE 8 TO CODE-CAR
      ELSE IF CAR = ")"
            MOVE 9 TO CODE-CAR
      ELSE IF CAR = """"
            MOVE 10 TO CODE-CAR
      ELSE IF CAR = ","
            MOVE 11 TO CODE-CAR
      ELSE IF CAR = ";"
            MOVE 12 TO CODE-CAR
      ELSE
            MOVE 7 TO CODE-CAR.


 TRAIT-DEBUT-LIGNE.
*********************
*
*    lorsque l'on sort de cette procedure, il y a 3 cas possibles
*    - soit le caractere lu est fin-de-ligne et POS = 0
*      ( dans le cas d'une ligne de commentaires )
*    - soit le caractere lu est "-" et pos = debut-c
*    - soit le caractere lu est le 1er de la zone A
*

      PERFORM LIRE-SUIVANT
      PERFORM LIRE-SUIVANT  UNTIL POS NOT < DEBUT-C.
      IF (POS = DEBUT-C) AND (CAR = "*" OR CAR = "/")
         PERFORM TRAIT-LIGNE-COMMENTAIRE
      ELSE IF (CAR = "-") AND (POS = DEBUT-C)
         PERFORM TRAIT-CONTI-LIGNE
      ELSE
         PERFORM LIRE-SUIVANT UNTIL POS NOT < DEBUT-A.

 TRAIT-LIGNE-COMMENTAIRE.
***************************
      PERFORM LIRE-SUIVANT UNTIL FIN-LIGNE = 1.

 TRAIT-CONTI-LIGNE.
*********************
      MOVE 13 TO CODE-CAR.
      PERFORM LIRE-SUIVANT.

*                                                                *
 LIRE-SUIVANT.
***************
*
*   lit le caractère suivant et calcul sa position sur la ligne
```

```
*
      IF FIN-LIGNE = 1
         MOVE 0 TO FIN-LIGNE
         MOVE 0 TO POS.
      READ FICHIER-ENTREE RECORD INTO CAR
           AT END GO TO RETOUR-TABLE.
      IF CAR = CAR-FIN-LIGNE
           READ FICHIER-ENTREE RECORD INTO CAR
                MOVE CAR-FIN-LIGNE TO CAR
                MOVE 1 TO FIN-LIGNE
                ADD 1  TO POS
      ELSE IF CAR = "       "
           ADD 8 TO POS
      ELSE
           ADD 1 TO POS.

* ================================================================ *
 GARDER-CAR.
************
*
*  va ajouter le caractère lu à MOT-LU
*  et ajouter 1 a la longueur de MOT-LU
*
      ADD 1 TO LONGUEUR.
      MOVE CAR TO CAR-MOT (LONGUEUR).

* ================================================================ *
 RETIRER-CAR.
*************
      MOVE " " TO CAR-MOT (LONGUEUR).
      SUBTRACT 1 FROM LONGUEUR.

* ================================================================ *
 ECRIRE-MOT.
************
*
*  va ecrire le mot-lu dans le fichier-resultat et
*  réinitialiser la longueur de MOT-LU à 0
*
      MOVE CODE-MOT TO CODE-SORTIE.
      WRITE REC-SORTIE.
      MOVE SPACES TO MOT-SORTIE.
      MOVE 0 TO LONGUEUR.


* ================================================================ *
 ATTRIBUER-CODE.
****************
*
*  lorque l'on entre dans cette procedure, mot-lu est lu en
*  entier et pret à être transmis. Le code de ce mot est
*  soit 10 (mot de titre), soit 20 (autre mot cobol).
*  Cette prodedure va modifier le code si le mot se trouve
*  dans sa liste, et sinon, ne modifiera pas le code.
*
*
      IF MOT-SORTIE = "PERFORM         "
      THEN MOVE 22  TO CODE-MOT
      ELSE IF MOT-SORTIE = "THRU          "
      THEN MOVE 23 TO CODE-MOT
      ELSE IF MOT-SORTIE = "UNTIL         "
      THEN MOVE 24 TO CODE-MOT
      ELSE IF MOT-SORTIE = "TIMES         "
      THEN MOVE 25 TO CODE-MOT
      ELSE IF MOT-SORTIE = "VARYING       "
```

```
      THEN MOVE 26 TO CODE-MOT
      ELSE IF MOT-SORTIE = "FROM          "
      THEN MOVE 27 TO CODE-MOT
      ELSE IF MOT-SORTIE = "BY            "
      THEN MOVE 28 TO CODE-MOT
      ELSE IF MOT-SORTIE = "AFTER         "
      THEN MOVE 29 TO CODE-MOT
      ELSE IF MOT-SORTIE = "IF            "
      THEN MOVE 30 TO CODE-MOT
      ELSE IF MOT-SORTIE = "ELSE          "
      THEN MOVE 31 TO CODE-MOT
      ELSE IF MOT-SORTIE = "SECTION       "
      THEN MOVE 32 TO CODE-MOT
      ELSE IF MOT-SORTIE = "OPEN          "
      THEN MOVE 33 TO CODE-MOT
      ELSE IF MOT-SORTIE = "CLOSE         "
      THEN MOVE 34 TO CODE-MOT
      ELSE IF MOT-SORTIE = "OUTPUT        "
      THEN MOVE 35 TO CODE-MOT
      ELSE IF MOT-SORTIE = "READ          "
      THEN MOVE 36 TO CODE-MOT
      ELSE IF MOT-SORTIE = "WRITE         "
      THEN MOVE 37 TO CODE-MOT
      ELSE IF MOT-SORTIE = "STOP          "
      THEN MOVE 38 TO CODE-MOT
      ELSE IF MOT-SORTIE = "RUN           "
      THEN MOVE 39 TO CODE-MOT
      ELSE IF MOT-SORTIE = "SELECT        "
      THEN MOVE 40 TO CODE-MOT
      ELSE IF MOT-SORTIE = "FD            "
      THEN MOVE 41 TO CODE-MOT.
```

```
*  ================================================================  *
*  ========    I M P L E M E N T A T I O N  ======================  *
*  ==========================  D E  ================================  *
*  ==========================  L A  ================================  *
*  =======================  T A B L E  =============================  *
*  ================================================================  *
```

```
 DEBUT-TABLE.
*************
      IF FIN-FICHIER = 1
         GO TO RETOUR-TABLE.
      GO TO ETAT1 ETAT2 ETAT3 ETAT4 ETAT5 ETAT6 ETAT7 ETAT8
            DEPENDING ON ETAT-PRESENT.
```

```
*  ================================================================  *
```

```
 ETAT1.
*******
      IF FIN-FICHIER = 1
         GO TO RETOUR-TABLE
      ELSE IF CODE-CAR = 1
          MOVE 4 TO ETAT-PRESENT
      ELSE IF CODE-CAR = 2
          MOVE 1 TO ETAT-PRESENT
      ELSE IF CODE-CAR = 3
          MOVE 2 TO ETAT-PRESENT
      ELSE IF CODE-CAR = 5
          MOVE 6 TO ETAT-PRESENT
      ELSE IF CODE-CAR = 6
```

```
           MOVE 6 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 7
           MOVE 1 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 8
           MOVE 7 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 10
           MOVE 8 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 11
           MOVE 5 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 12
           MOVE 1 TO ETAT-PRESENT.

       GO TO ACTION1  ACTION4  ACTION12 ACTION13 ACTION1  ACTION1
             ACTION21 ACTION6  ACTIONO  ACTION1  ACTION1  ACTION4
             ACTIONO DEPENDING ON CODE-CAR.

 ETAT2.
*******

       IF CODE-CAR = 1
           MOVE 1 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 11
           MOVE 1 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 12
           MOVE 1 TO ETAT-PRESENT.

       GO TO ACTION2  ACTION18 ACTION1  ACTION1  ACTIONO  ACTION1
             ACTIONO  ACTION20 ACTION1  ACTIONO  ACTION2  ACTION2
             ACTION16 DEPENDING ON CODE-CAR.


 ETAT3.
*******

       IF CODE-CAR = 2
           MOVE 1 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 3
           MOVE 2 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 6
           MOVE 2 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 8
           MOVE 2 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 12
           MOVE 1 TO ETAT-PRESENT.

       GO TO ACTION11 ACTION14 ACTION1  ACTION1  ACTIONO  ACTION1
             ACTIONO  ACTION20 ACTIONO  ACTIONO  ACTION5  ACTION14
             ACTION16 DEPENDING ON CODE-CAR.

 ETAT4.
*******

       IF CODE-CAR = 2
           MOVE 1 TO ETAT-PRESENT
       ELSE IF CODE-CAR = 4
           MOVE 3 TO ETAT-PRESENT.

       GO TO ACTIONO  ACTION3  ACTIONO  ACTION1  ACTIONO  ACTIONO
             ACTIONO  ACTIONO  ACTIONO  ACTIONO  ACTIONO  ACTIONO
             ACTION16 DEPENDING ON CODE-CAR.

 ETAT5.
*******

       IF CODE-CAR =  2
           MOVE 1 TO ETAT-PRESENT
```

```
     ELSE IF CODE-CAR = 4
         MOVE 3 TO ETAT-PRESENT.

     GO TO ACTIONO  ACTION15 ACTIONO  ACTION1  ACTIONO  ACTIONO
           ACTIONO  ACTIONO  ACTIONO  ACTIONO  ACTIONO  ACTIONO
           ACTION16 DEPENDING ON CODE-CAR.

 ETAT6.
*******

     IF CODE-CAR =  1
         MOVE 3 TO ETAT-PRESENT
     ELSE IF CODE-CAR = 2
         MOVE 1 TO ETAT-PRESENT
     ELSE IF CODE-CAR = 4
         MOVE 3 TO ETAT-PRESENT.

     GO TO ACTION1  ACTION10 ACTIONO  ACTION1  ACTIONO  ACTIONO
           ACTIONO  ACTIONO  ACTIONO  ACTIONO  ACTIONO  ACTIONO
           ACTION16 DEPENDING ON CODE-CAR.

 ETAT7.
*******


     GO TO ACTION1  ACTION17 ACTION1  ACTION1  ACTION1  ACTION1
           ACTION1  ACTION7  ACTION8  ACTION1  ACTION1  ACTION1
           ACTION16 DEPENDING ON CODE-CAR.
 ETAT8.
*******


     GO TO ACTION1  ACTION22 ACTION1  ACTION1  ACTION1  ACTION1
           ACTION1  ACTION1  ACTION1  ACTION9  ACTION1  ACTION1
           ACTION19 DEPENDING ON CODE-CAR.

* ================================================================ *
 ACTIONO.
*********
     DISPLAY "ERREUR DE COMPILATION" UPON CONSOLE
     DISPLAY "CAR:" CAR "code-car:" CODE-CAR ":" MOT-SORTIE
                              "ETAT :" ETAT-PRESENT UPON CONSOLE

     GO TO RETOUR-TABLE.

 ACTION1.
*********
     PERFORM GARDER-CAR
     PERFORM LIRE-CAR.
     GO TO DEBUT-TABLE.


 ACTION2.
*********
     PERFORM ATTRIBUER-CODE
     PERFORM ECRIRE-MOT
     GO TO DEBUT-TABLE.


 ACTION3.
*********
     MOVE 1 TO CODE-MOT
     PERFORM ECRIRE-MOT
     PERFORM LIRE-CAR.
     GO TO DEBUT-TABLE.
```

```
 ACTION4.
*********
      PERFORM LIRE-CAR.
      GO TO DEBUT-TABLE.


 ACTION5.
*********
      PERFORM GARDER-CAR
      PERFORM LIRE-CAR
      IF CAR = " "
         PERFORM RETIRER-CAR
         MOVE 2 TO CODE-MOT
         PERFORM ECRIRE-MOT
         PERFORM LIRE-CAR
         MOVE 1 TO ETAT-PRESENT.
      GO TO DEBUT-TABLE.


 ACTION6.
*********
      PERFORM GARDER-CAR
      MOVE 1 TO COMPT-PARENTHESE
      PERFORM LIRE-CAR
      GO TO DEBUT-TABLE.

 ACTION7.
*********
      PERFORM GARDER-CAR
      ADD 1 TO COMPT-PARENTHESE
      PERFORM LIRE-CAR
      GO TO DEBUT-TABLE.


 ACTION8.
*********
      PERFORM GARDER-CAR
      SUBTRACT 1 FROM COMPT-PARENTHESE
      IF COMPT-PARENTHESE = 0
         MOVE 4 TO CODE-MOT
         PERFORM ECRIRE-MOT
         PERFORM LIRE-CAR
         MOVE 1 TO ETAT-PRESENT
      ELSE
         PERFORM LIRE-CAR
         MOVE 7 TO ETAT-PRESENT.
      GO TO DEBUT-TABLE.

 ACTION9.
*********
      PERFORM GARDER-CAR
      PERFORM LIRE-CAR
      IF CODE-CAR = 10
         PERFORM GARDER-CAR
         PERFORM LIRE-CAR
      ELSE
         MOVE 3 TO CODE-MOT
         PERFORM ECRIRE-MOT
         MOVE 1 TO ETAT-PRESENT.
      GO TO DEBUT-TABLE.


 ACTION10.
*********
      MOVE 5 TO CODE-CAR
      PERFORM ECRIRE-MOT
```

```
           PERFORM LIRE-CAR
           GO TO DEBUT-TABLE.


       ACTION11.
      *********
           PERFORM GARDER-CAR
           PERFORM LIRE-CAR
           IF CAR = " "
              PERFORM RETIRER-CAR
              MOVE 2 TO CODE-MOT
              PERFORM ECRIRE-MOT
              MOVE "." TO CAR
              PERFORM GARDER-CAR
              MOVE 1 TO CODE-MOT
              PERFORM ECRIRE-MOT
              PERFORM LIRE-CAR
              MOVE 1 TO ETAT-PRESENT
           ELSE IF CODE-CAR = 4
              PERFORM GARDER-CAR
              PERFORM LIRE-CAR
              MOVE 3 TO ETAT-PRESENT.
           GO TO DEBUT-TABLE.


       ACTION12.
      *********
           IF POS < DEBUT-B
              MOVE 10 TO CODE-MOT
           ELSE
              MOVE 20 TO CODE-MOT.
           PERFORM GARDER-CAR
           PERFORM LIRE-CAR.
           GO TO DEBUT-TABLE.


       ACTION13.
      *********
           IF POS < DEBUT-B
              MOVE 10 TO CODE-MOT
              PERFORM GARDER-CAR
              PERFORM LIRE-CAR
              MOVE 2 TO ETAT-PRESENT
           ELSE
              PERFORM GARDER-CAR
              PERFORM LIRE-CAR
              MOVE 3 TO ETAT-PRESENT.
           GO TO DEBUT-TABLE.

       ACTION14.
      *********
           PERFORM LIRE-CAR UNTIL CODE-CAR NOT = 2
           IF CODE-CAR = 13
              PERFORM LIRE-CAR UNTIL CODE-CAR NOT = 2
              MOVE 3 TO ETAT-PRESENT
           ELSE
              MOVE 2 TO CODE-MOT
              PERFORM ECRIRE-MOT
              MOVE 1 TO ETAT-PRESENT.
           GO TO DEBUT-TABLE.

       ACTION15.
      *********
           PERFORM RETIRER-CAR
           PERFORM LIRE-CAR
           GO TO DEBUT-TABLE.
```

```
 ACTION16.
*********
      PERFORM LIRE-CAR
      PERFORM LIRE-CAR UNTIL CODE-CAR NOT = 2
      GO TO DEBUT-TABLE.

 ACTION17.
*********
      MOVE " " TO CAR
      PERFORM GARDER-CAR
      PERFORM LIRE-CAR UNTIL CODE-CAR NOT = 2
      GO TO DEBUT-TABLE.


 ACTION18.
*********
      PERFORM LIRE-CAR UNTIL CODE-CAR NOT = 2
      IF CODE-CAR = 13
         PERFORM LIRE-CAR UNTIL CODE-CAR NOT = 2
         MOVE 2 TO ETAT-PRESENT
      ELSE
         PERFORM ATTRIBUER-CODE
         PERFORM ECRIRE-MOT
         MOVE 1 TO ETAT-PRESENT.
      GO TO DEBUT-TABLE.


 ACTION19.
*********
      PERFORM LIRE-CAR UNTIL CODE-CAR  = 10
      PERFORM LIRE-CAR
      GO TO DEBUT-TABLE.

 ACTION20.
*********
      PERFORM GARDER-CAR
      MOVE 21 TO CODE-MOT
      PERFORM LIRE-CAR.
      GO TO DEBUT-TABLE.

 ACTION21.
*********
      PERFORM GARDER-CAR
      MOVE 5 TO CODE-CAR
      PERFORM ECRIRE-MOT
      PERFORM LIRE-CAR
      GO TO DEBUT-TABLE.

 ACTION22.
*********
      IF CAR = " " OR CAR = "      "
         PERFORM GARDER-CAR
      ELSE
         MOVE " " TO CAR
         COMPUTE NBRE-ESPACE = FIN-B - POS
         PERFORM GARDER-CAR  NBRE-ESPACE TIMES.
      PERFORM LIRE-CAR.
      GO TO DEBUT-TABLE.

*
*
*              FIN DU PROGRAMME
*             ANALYSEUR LEXICAL
*
```

```
*
*
*       RE-CONSTRUCTEUR DE TEXTE COBOL
*       A PARTIR DE MOTS
*
*
 IDENTIFICATION DIVISION.
*************************
 PROGRAM-ID.  CONSTRUCTEUR.


 ENVIRONMENT DIVISION.
***********************
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 SELECT FICHIER-ENTREE  ASSIGN TO FICHIER-SOURCE
        ORGANIZATION IS SEQUENTIAL.

 SELECT FICHIER-SORTIE ASSIGN TO FICHIER-RESULTAT
        ORGANIZATION IS SEQUENTIAL.


 DATA DIVISION.
****************

 FILE SECTION.
***************

 FD FICHIER-ENTREE LABEL RECORD IS STANDARD.
 01 REC-ENTREE.
    02 CODE-MOT       PIC 99.
    02 LONGUEUR       PIC 999.
    02 MOT-RECU.
       03 CAR-MOT PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONGUEUR.

 FD FICHIER-SORTIE LABEL RECORD IS STANDARD.
 01 CAR-FICHIER   PIC X.

 WORKING-STORAGE SECTION.
**************************
 01 DEBUT-A    PIC 99 VALUE 8.
 01 DEBUT-B    PIC 99 VALUE 12.
 01 FIN-B      PIC 99 VALUE 72.
 01 DEBUT-C    PIC 99 VALUE 7.

 01 FICHIER-SOURCE    PIC X(12).
 01 FICHIER-RESULTAT  PIC X(12).

 01 POS-LIGNE       PIC 99.
 01 POS-LETTRE      PIC 999.

 01 FIN-LIGNE       PIC 9.
 01 ETAT-PRESENT    PIC 99.

 01 FIN-MOT         PIC 9.
 01 FIN-FICHIER     PIC 9.
 01 FIN-LI PIC 99 COMP VALUE 13.
 01 CAR-FIN-LIGNE1 REDEFINES FIN-LI PIC X.
 01 CAR-FIN-LIGNE2 PIC X VALUE "
".


 01 NBRE-CAR        PIC 99.
```

```
 PROCEDURE DIVISION.
**********************
*
*    SAISIE    DES PARAMETRES
*
      DISPLAY "entrez le fichier source (.MOT)" UPON CONSOLE.
      ACCEPT FICHIER-SOURCE FROM CONSOLE.
      MOVE FICHIER-SOURCE TO FICHIER-RESULTAT.
      INSPECT FICHIER-SOURCE REPLACING FIRST "    " BY ".MOT".
      INSPECT FICHIER-RESULTAT REPLACING FIRST "     "
                                       BY "C.CBL".

      DISPLAY "le fichier source est " FICHIER-SOURCE
              "et le fichier resultat est " FICHIER-RESULTAT
                                              UPON CONSOLE.

*
*
*           DEBUT DU PROGRAMME
*
*
      OPEN INPUT FICHIER-ENTREE.
      OPEN OUTPUT FICHIER-SORTIE.
      MOVE O TO FIN-FICHIER.
      MOVE " " TO CAR-FICHIER.
      PERFORM ECRIRE-CAR DEBUT-A TIMES.
      MOVE DEBUT-A TO POS-LIGNE.
      READ FICHIER-ENTREE RECORD
           AT END MOVE 1 TO FIN-FICHIER.

      PERFORM TRAIT-MOT UNTIL ( FIN-FICHIER = 1 ) .

      CLOSE FICHIER-ENTREE.
      CLOSE FICHIER-SORTIE.
      STOP RUN.
* ================================================================ *
 ECRIRE-CAR.
      WRITE CAR-FICHIER.
      ADD 1 TO POS-LIGNE.

*

 TRAIT-MOT.
***********
      IF ( CODE-MOT = 10 )
*     si mot de titre
         PERFORM ECRIRE-FIN-LIGNE
         MOVE SPACE TO CAR-FICHIER
         COMPUTE NBRE-CAR  = DEBUT-A - 1
         PERFORM ECRIRE-CAR NBRE-CAR  TIMES
         PERFORM ECRIRE-MOT
      ELSE IF (FIN-B < LONGUEUR + POS-LIGNE)
*          si il n'y a pas assez de place sur la ligne
         IF   (LONGUEUR NOT > FIN-B - DEBUT-B)
*             si le mot est plus petit que la longueur d'une ligne
            PERFORM ECRIRE-FIN-LIGNE
            MOVE SPACE TO CAR-FICHIER
            COMPUTE NBRE-CAR  = DEBUT-B - 1
            PERFORM ECRIRE-CAR NBRE-CAR  TIMES
            PERFORM ECRIRE-MOT
        ELSE
*           si le mot est plus grand que la longueur d'une ligne
            PERFORM ECRIRE-LETTRE UNTIL (POS-LIGNE = FIN-B)
            PERFORM TRAIT-GRAND-MOT UNTIL (FIN-MOT = 1)
            MOVE O TO FIN-MOT
            MOVE SPACE TO CAR-FICHIER
```

```
            PERFORM ECRIRE-CAR
      ELSE
*        si il y a assez de place sur la ligne
         PERFORM ECRIRE-MOT.
      IF (CODE-MOT = 1)
*     si le mot est un point
            PERFORM ECRIRE-FIN-LIGNE
            MOVE SPACE TO CAR-FICHIER
            COMPUTE NBRE-CAR  = DEBUT-B - 1
            PERFORM ECRIRE-CAR NBRE-CAR  TIMES.
      READ FICHIER-ENTREE RECORD
           AT END MOVE 1 TO FIN-FICHIER.
      MOVE O TO POS-LETTRE.



* ================================================================ *
 TRAIT-GRAND-MOT.
           PERFORM ECRIRE-FIN-LIGNE
           MOVE SPACE TO CAR-FICHIER
           COMPUTE NBRE-CAR  = DEBUT-C - 1
           PERFORM ECRIRE-CAR NBRE-CAR  TIMES
           MOVE "-" TO CAR-FICHIER
           PERFORM ECRIRE-CAR
           MOVE SPACE TO CAR-FICHIER
           COMPUTE NBRE-CAR    = DEBUT-B - DEBUT-C - 1
           PERFORM ECRIRE-CAR NBRE-CAR    TIMES
           IF (CODE-MOT = 3)
              MOVE """" TO CAR-FICHIER
              PERFORM ECRIRE-CAR
           ELSE NEXT SENTENCE.
           PERFORM ECRIRE-LETTRE UNTIL (FIN-MOT = 1).

 ECRIRE-MOT.
      PERFORM ECRIRE-LETTRE UNTIL (FIN-MOT = 1).
      MOVE O TO FIN-MOT.
      MOVE " " TO CAR-FICHIER.
      PERFORM ECRIRE-CAR.

 ECRIRE-LETTRE.
      ADD 1 TO POS-LETTRE
      MOVE CAR-MOT (POS-LETTRE) TO CAR-FICHIER
      PERFORM ECRIRE-CAR
      IF (POS-LETTRE = LONGUEUR)
         MOVE 1 TO FIN-MOT.

 ECRIRE-FIN-LIGNE.

         MOVE CAR-FIN-LIGNE1 TO CAR-FICHIER
         PERFORM ECRIRE-CAR
         MOVE CAR-FIN-LIGNE2 TO CAR-FICHIER
         PERFORM ECRIRE-CAR.
         MOVE O TO POS-LIGNE.
```

```
*
*          TRANSFORMATION DES PERFORM ET DES IF
*                     PREMIERE PASSE
*
*
*

 IDENTIFICATION DIVISION.
*************************
 PROGRAM-ID.  PER-1-PASS.


 ENVIRONMENT DIVISION.
***********************
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 SELECT FICHIER-ENTREE  ASSIGN TO FICHIER-SOURCE
        ORGANIZATION IS SEQUENTIAL.

 SELECT FICHIER-SORTIE ASSIGN TO FICHIER-RESULTAT
        ORGANIZATION IS SEQUENTIAL.

 SELECT TABLE-TITRE ASSIGN TO "TABLE.PER"
        ORGANIZATION IS INDEXED
        ACCESS MODE IS DYNAMIC
        RECORD KEY IS NOM-TITRE.


 DATA DIVISION.
****************

 FILE SECTION.
***************

 FD FICHIER-ENTREE LABEL RECORD IS STANDARD.
 01 REC-ENTREE.
    02 CODE-ENT       PIC 99.
    02 LONG-ENT       PIC 999.
    02 MOT-ENT.
       03 CAR-ENT PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-ENT.

 FD FICHIER-SORTIE LABEL RECORD IS STANDARD.
 01 REC-SORTIE.
    02 CODE-SOR       PIC 99.
    02 LONG-SOR       PIC 999.
    02 MOT-SOR.
       03 CAR-SOR PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-SOR.

 FD TABLE-TITRE LABEL RECORD IS STANDARD.
 01 REC-TITRE.
    02 NOM-TITRE PIC X(30).
    02 NBRE-APPEL PIC 99.


 WORKING-STORAGE SECTION.
**************************

 01 DEBUT-A    PIC 99 VALUE 8.
 01 DEBUT-B    PIC 99 VALUE 12.
 01 FIN-B      PIC 99 VALUE 72.
 01 DEBUT-C    PIC 99 VALUE 7.

 01 FICHIER-SOURCE     PIC X(12).
```

```
01 FICHIER-RESULTAT  PIC X(12).

01 COMPT-PARENTHESE   PIC 99.
01 NBRE-ESPACE        PIC 99.
01 FIN-LIGNE          PIC 9.
01 ETAT-PRESENT   PIC 99.
01 CAS                PIC 99.
01 POINTEUR           PIC 999.

01 TROUVE PIC 9.
01 TROUVE-TITRE PIC 9.
01 TROUVE-NUMIF PIC 9.

01 FIN-FICHIER    PIC 9.

01 BIDON PIC X.
01 COMPT  PIC 9(5).

01 REC-PROC.
   02 LONG-PROC     PIC 999.
   02 PROC.
      03 CAR-PROC PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-PROC.

01 REC-PROC1.
   02 CODE-PROC1     PIC 99.
   02 LONG-PROC1     PIC 999.
   02 PROC1.
      03 CAR-PROC1 PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-PROC1.

01 REC-COND.
   02 CODE-COND      PIC 99.
   02 LONG-COND     PIC 999.
   02 COND.
      03 CAR-COND PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-COND.

01 REC-NOMBRE.
   02 CODE-NOMBRE    PIC 99.
   02 LONG-NOMBRE   PIC 999.
   02 NOMBRE.
      03 CAR-NOMBRE PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-NOMBRE.

01 REC-COND1.
   02 CODE-COND1     PIC 99.
   02 LONG-COND1     PIC 999.
   02 COND1.
      03 CAR-COND1 PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-COND1.

01 REC-COND2.
   02 CODE-COND2     PIC 99.
   02 LONG-COND2     PIC 999.
   02 COND2.
      03 CAR-COND2 PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-COND2.

01 REC-COND3.
   02 CODE-COND3     PIC 99.
   02 LONG-COND3     PIC 999.
   02 COND3.
      03 CAR-COND3 PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-COND3.
```

```
01 REC-INIT1.
   02 CODE-INIT1     PIC 99.
   02 LONG-INIT1     PIC 999.
   02 INIT1.
      03 CAR-INIT1 PIC X OCCURS 1 TO 255 TIMES
                         DEPENDING ON LONG-INIT1.

01 REC-INIT2.
   02 CODE-INIT2     PIC 99.
   02 LONG-INIT2     PIC 999.
   02 INIT2.
      03 CAR-INIT2 PIC X OCCURS 1 TO 255 TIMES
                         DEPENDING ON LONG-INIT2.

01 REC-INIT3.
   02 CODE-INIT3     PIC 99.
   02 LONG-INIT3     PIC 999.
   02 INIT3.
      03 CAR-INIT3 PIC X OCCURS 1 TO 255 TIMES
                         DEPENDING ON LONG-INIT3.

01 REC-INCR1.
   02 CODE-INCR1     PIC 99.
   02 LONG-INCR1     PIC 999.
   02 INCR1.
      03 CAR-INCR1 PIC X OCCURS 1 TO 255 TIMES
                         DEPENDING ON LONG-INCR1.


01 REC-INCR2.
   02 CODE-INCR2     PIC 99.
   02 LONG-INCR2     PIC 999.
   02 INCR2.
      03 CAR-INCR2 PIC X OCCURS 1 TO 255 TIMES
                         DEPENDING ON LONG-INCR2.

01 REC-INCR3.
   02 CODE-INCR3     PIC 99.
   02 LONG-INCR3     PIC 999.
   02 INCR3.
      03 CAR-INCR3 PIC X OCCURS 1 TO 255 TIMES
                         DEPENDING ON LONG-INCR3.

01 REC-VAR1.
   02 CODE-VAR1      PIC 99.
   02 LONG-VAR1     PIC 999.
   02 VAR1.
      03 CAR-VAR1 PIC X OCCURS 1 TO 255 TIMES
                         DEPENDING ON LONG-VAR1.

01 REC-VAR2.
   02 CODE-VAR2      PIC 99.
   02 LONG-VAR2     PIC 999.
   02 VAR2.
      03 CAR-VAR2 PIC X OCCURS 1 TO 255 TIMES
                         DEPENDING ON LONG-VAR2.

01 REC-VAR3.
   02 CODE-VAR3      PIC 99.
   02 LONG-VAR3     PIC 999.
   02 VAR3.
      03 CAR-VAR3 PIC X OCCURS 1 TO 255 TIMES
                         DEPENDING ON LONG-VAR3.


01 NBRE-TIME PIC 99.
```

```
 01 NBRE-VAR  PIC 99.
 01 NOMBRE-IF-DEBUT PIC 99.
 01 NOMBRE-IF PIC 99.
 01 IF-EN-COURS PIC 99.

 01 PILE-IF.
    02 PILE OCCURS 20.
       03 NUM-IF PIC 99.
       03 IF-APPARIE PIC 9.
    02 POINTEUR-PILE PIC 99.
    02 POINTEUR-PILE-2 PIC 99.


 PROCEDURE DIVISION.
**********************
 PRINCIPAL SECTION.
**********************
*  ================================================================ *
*  ============== S A I S I E  D E S  P A R A M E T R E S ========== *
*  ================================================================ *

     DISPLAY "entrez le fichier source (.MOT)" UPON CONSOLE.
     ACCEPT FICHIER-SOURCE FROM CONSOLE.
     MOVE FICHIER-SOURCE TO FICHIER-RESULTAT.
     INSPECT FICHIER-SOURCE REPLACING FIRST "    " BY ".MOT".
     INSPECT FICHIER-RESULTAT REPLACING FIRST "    " BY "P.P1".

     DISPLAY "le fichier source est " FICHIER-SOURCE
             "et le fichier resultat est " FICHIER-RESULTAT
                                           UPON CONSOLE.

*  ================================================================ *
*  ============== D E B U T  D U  P R O G R A M M E ============== *
*  ================================================================ *

     OPEN INPUT FICHIER-ENTREE.
     OPEN OUTPUT FICHIER-SORTIE.
     OPEN OUTPUT TABLE-TITRE.
     CLOSE TABLE-TITRE.
     OPEN I-O TABLE-TITRE.
     MOVE 0 TO NBRE-TIME.
     MOVE 0 TO NBRE-VAR.

     PERFORM LIRE-SUIVANT.


     PERFORM PASSE-1 UNTIL FIN-FICHIER = 1.


     MOVE "NOMBRE-TIME" TO NOM-TITRE.
     MOVE NBRE-TIME TO NBRE-APPEL.
     WRITE REC-TITRE INVALID KEY GO TO ERREUR-GRAVE.



     CLOSE FICHIER-ENTREE.
     CLOSE FICHIER-SORTIE.
     CLOSE TABLE-TITRE.
     STOP RUN.

*  ================================================================ *
*  ================  P R E M I E R E  P A S S E  ================== *
*  ================================================================ *

 PASSE-1.
*********
```

```
*********
     IF CODE-ENT = 22                                                  perfor
        PERFORM TRAIT-PERF
        PERFORM ELIMINE-POINT
     ELSE IF CODE-ENT = 30                                             if
        PERFORM TRAIT-IF
     ELSE                                                              autre
        PERFORM ECRIRE-MEME-MOT
        PERFORM LIRE-SUIVANT.


*    ═══════════════════════════════════════════════════════════════ *
* ═══════════════ P R I M I T I V E S ═══════════════════════════ *
*    ═══════════════════════════════════════════════════════════════ *

 LIRE-SUIVANT.
***************
     READ FICHIER-ENTREE AT END MOVE 1 TO FIN-FICHIER.

 ECRIRE-MOT.
*************
     WRITE REC-SORTIE.

 ECRIRE-MEME-MOT.
*************
     WRITE REC-SORTIE FROM REC-ENTREE.

 ELIMINE-POINT.
*************
     IF ((CODE-ENT = 1) AND (CODE-SOR = 1))
        PERFORM LIRE-SUIVANT.

 ECRIRE-NBRE.
*************
     WRITE REC-SORTIE FROM REC-NOMBRE.

 LIRE-SUIVANT-VAR1.
****************
     MOVE LONG-ENT TO LONG-VAR1.
     MOVE REC-ENTREE TO REC-VAR1.
     PERFORM LIRE-SUIVANT.

 LIRE-SUIVANT-VAR2.
****************
     MOVE LONG-ENT TO LONG-VAR2.
     MOVE REC-ENTREE TO REC-VAR2.
     PERFORM LIRE-SUIVANT.

 LIRE-SUIVANT-VAR3.
****************
     MOVE LONG-ENT TO LONG-VAR3.
     MOVE REC-ENTREE TO REC-VAR3.
     PERFORM LIRE-SUIVANT.

 LIRE-SUIVANT-INIT1.
****************
     MOVE LONG-ENT TO LONG-INIT1.
     MOVE REC-ENTREE TO REC-INIT1
     PERFORM LIRE-SUIVANT.

 LIRE-SUIVANT-INIT2.
****************
     MOVE LONG-ENT TO LONG-INIT2.
     MOVE REC-ENTREE TO REC-INIT2.
     PERFORM LIRE-SUIVANT.
```

```
 LIRE-SUIVANT-INIT3.
***************
      MOVE LONG-ENT TO LONG-INIT3.
      MOVE REC-ENTREE TO REC-INIT3
      PERFORM LIRE-SUIVANT.

 LIRE-SUIVANT-INCR1.
***************
      MOVE LONG-ENT TO LONG-INCR1.
      MOVE REC-ENTREE TO REC-INCR1
      PERFORM LIRE-SUIVANT.

 LIRE-SUIVANT-INCR2.
***************
      MOVE LONG-ENT TO LONG-INCR2.
      MOVE REC-ENTREE TO REC-INCR2
      PERFORM LIRE-SUIVANT.

 LIRE-SUIVANT-INCR3.
***************
      MOVE LONG-ENT TO LONG-INCR3.
      MOVE REC-ENTREE TO REC-INCR3
      PERFORM LIRE-SUIVANT.

 AJOUTER-NBRE.
**************  ajouter rec-entree a nombre.

      MOVE LONG-NOMBRE TO POINTEUR.
      STRING  MOT-ENT DELIMITED BY SIZE INTO NOMBRE
      WITH POINTER POINTEUR.



 AJOUTER-TITRE.
*
*  si proc se trouve dans le fichier, va incrémenter nbre-appel
*  de 1 , et si proc ne se trouve pas dans la table ,   va l'y
*  ajouter, et mettre nbre-appel à 1.
*


      PERFORM CHERCHER-TITRE.
      IF TROUVE-TITRE = 1
         ADD 1 TO NBRE-APPEL
         REWRITE REC-TITRE
      ELSE
         MOVE PROC TO NOM-TITRE
         MOVE 1 TO NBRE-APPEL
         WRITE REC-TITRE INVALID KEY GO TO ERREUR-GRAVE.



 CHERCHER-TITRE.
*
*  cherche si proc se trouve dans le fichier table-titre.
*  si oui, donne sa position et met trouve-titre = 1
*  si non, met trouve-titre = 0
*
      MOVE 1 TO TROUVE-TITRE.
      MOVE PROC TO NOM-TITRE.
      READ TABLE-TITRE RECORD
           KEY IS NOM-TITRE
           INVALID KEY MOVE 0 TO TROUVE-TITRE.


*                                                                *
```

```
*  ═══════   I M P L E M E N T A T I O N ═══════════════════ *
*  ═══════  DES ETAPES DE LA TRANSFORMATION  ═══════════════ *
*  ═══════════════  DES PERFORM  ═══════════════════════════ *
*  ═══════  note : étape est noté ici état  ═══════════════ *
*  ═══════════════════════════════════════════════════════ *

 TRAIT-PERF SECTION.
*
*
      MOVE 1 TO ETAT-PRESENT.
      PERFORM LIRE-SUIVANT.

 DEBUT-TABLE-PERF.
******************
      IF FIN-FICHIER = 1
         GO TO FIN-TABLE-PERF.
      GO TO ETAT1 ETAT2 ETAT3 ETAT4 ETAT5 ETAT6
            DEPENDING ON ETAT-PRESENT.

* ═══════════════════════════════════════════════════════ *


 ETAT0.                                                    ETAT FI
*******                                                    NAL
      IF CAS = 1
         PERFORM CAS-1
      ELSE IF CAS = 2
         PERFORM CAS-2
      ELSE IF CAS = 3
         PERFORM CAS-3
      ELSE IF CAS = 4
         PERFORM CAS-4
      ELSE IF CAS = 5
         PERFORM CAS-5
      ELSE IF CAS = 6
         PERFORM CAS-6
      ELSE IF CAS = 11
         PERFORM CAS-1
         PERFORM ECRIRE-NBRE.
      GO TO FIN-TABLE-PERF.

 ETAT1.
*******
      IF CODE-ENT = 20                                     MOT
          MOVE 2 TO ETAT-PRESENT                           COBOL
          PERFORM ACTION1
      ELSE                                                 ERREUR
          PERFORM ACTIONE.
      GO TO DEBUT-TABLE-PERF.

 ETAT2.
*******
      IF CODE-ENT NOT = 23                                 THRU
         PERFORM ACTION12.


      IF CODE-ENT = 2                                      NOMBRE
          MOVE 5 TO ETAT-PRESENT
          PERFORM ACTION7
      ELSE IF CODE-ENT = 20                                MOT
          MOVE 6 TO ETAT-PRESENT
          PERFORM ACTION7
      ELSE IF CODE-ENT = 23                                THRU
          MOVE 2 TO ETAT-PRESENT
          PERFORM ACTION2
      ELSE IF CODE-ENT = 24                                UNTIL
```

```
           MOVE 0 TO ETAT-PRESENT
           PERFORM ACTION3
      ELSE IF CODE-ENT = 26                                          VARYIN
           MOVE 3 TO ETAT-PRESENT
           PERFORM ACTION4
      ELSE                                                           AUTRE
           MOVE 0 TO ETAT-PRESENT
           PERFORM ACTION11.
      GO TO DEBUT-TABLE-PERF.


 ETAT3.
*******
      IF CODE-ENT = 29
           MOVE 4 TO ETAT-PRESENT
           PERFORM ACTION5
      ELSE
           MOVE 0 TO ETAT-PRESENT
           PERFORM ACTIONE.
      GO TO DEBUT-TABLE-PERF.


 ETAT4.
*******
      IF CODE-ENT = 29
           MOVE 0 TO ETAT-PRESENT
           PERFORM ACTION6
      ELSE
           MOVE 0 TO ETAT-PRESENT
           PERFORM ACTIONE.
      GO TO DEBUT-TABLE-PERF.



 ETAT5.
*******
      IF CODE-ENT = 25
           MOVE 0 TO ETAT-PRESENT
           PERFORM ACTION8
      ELSE
           MOVE 0 TO ETAT-PRESENT
           PERFORM ACTION9.
      GO TO DEBUT-TABLE-PERF.

 ETAT6.
*******
      IF CODE-ENT = 25
           MOVE 0 TO ETAT-PRESENT
           PERFORM ACTION8
      ELSE IF CODE-ENT = 4
           PERFORM ACTION10
      ELSE
           MOVE 0 TO ETAT-PRESENT
           PERFORM ACTION9.
      GO TO DEBUT-TABLE-PERF.



* ═══════════════════════════════════════════════════════════════ *
 ACTIONE.
*********


 ACTION1.
*********
     MOVE LONG-ENT TO LONG-PROC1.
```

```
      MOVE MOT-ENT TO PROC1.
      MOVE LONG-ENT TO LONG-PROC.
      MOVE MOT-ENT TO PROC.
      PERFORM LIRE-SUIVANT.

 ACTION2.
*********
      MOVE LONG-ENT TO LONG-PROC.
      MOVE MOT-ENT TO PROC.
      PERFORM LIRE-SUIVANT.


 ACTION3.
*********
      PERFORM LIRE-SUIVANT.
      MOVE LONG-ENT TO LONG-COND.
      MOVE MOT-ENT TO COND.
      MOVE 2 TO CAS.
      PERFORM LIRE-SUIVANT.




 ACTION4.
*********
      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT-VAR1 UNTIL CODE-ENT = 27.
      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT-INIT1 UNTIL CODE-ENT = 28.
      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT-INCR1 UNTIL CODE-ENT = 24.
      PERFORM LIRE-SUIVANT.
      MOVE LONG-ENT TO LONG-COND1.
      MOVE MOT-ENT TO COND1.
      PERFORM LIRE-SUIVANT.
      IF CODE-ENT = 1
         PERFORM LIRE-SUIVANT.
      ADD 1 TO NBRE-VAR.
      MOVE 4 TO CAS.


 ACTION5.
*********
      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT-VAR2 UNTIL CODE-ENT = 27.
      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT-INIT2 UNTIL CODE-ENT = 28.
      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT-INCR2 UNTIL CODE-ENT = 24.
      PERFORM LIRE-SUIVANT.
      MOVE LONG-ENT TO LONG-COND2.
      MOVE MOT-ENT TO COND2.
      PERFORM LIRE-SUIVANT.
      IF CODE-ENT = 1
         PERFORM LIRE-SUIVANT.
      MOVE 5 TO CAS.



 ACTION6.
*********
      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT-VAR3 UNTIL CODE-ENT = 27.
      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT-INIT3 UNTIL CODE-ENT = 28.
```

```
      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT-INCR3 UNTIL CODE-ENT = 24.
      PERFORM LIRE-SUIVANT.
      MOVE LONG-ENT TO LONG-COND3.
      MOVE MOT-ENT TO COND3.
      PERFORM LIRE-SUIVANT.
      IF CODE-ENT = 1
         PERFORM LIRE-SUIVANT.
      MOVE 6 TO CAS.


 ACTION7.
*********
      MOVE LONG-ENT TO LONG-NOMBRE.
      MOVE MOT-ENT TO NOMBRE.
      PERFORM LIRE-SUIVANT.


 ACTION8.
*********
      PERFORM LIRE-SUIVANT.
      ADD 1 TO NBRE-TIME.

      MOVE 3 TO CAS.

 ACTION9.
*********
      MOVE 11 TO CAS.

 ACTION10.
*********
      PERFORM AJOUTER-NBRE.
      PERFORM LIRE-SUIVANT.
      IF CODE-ENT = 25
         MOVE 3 TO CAS
      ELSE
         MOVE 11 TO CAS.

 ACTION11.
*********
      MOVE 1 TO CAS.

 ACTION12.
*********
      PERFORM AJOUTER-TITRE.

* ═══════════════════════════════════════════════════════════════ *
* ═══════    ECRITURE DES MOTS COBOL DANS ═══════════════════ *
* ═══════════ LE FICHIER DE SORTIE          ═══════════════ *
* ═══════════════════════════════════════════════════════════════ *

 CAS-1.
******
*
*    ╔═══════════════════════════════════════╗
*    ║ Correspond au point 2.2.3.1.1         ║
*    ╚═══════════════════════════════════════╝

*     MOVE 20 TO CODE-SOR.
*     MOVE 4 TO LONG-MOT.
*     MOVE "MOVE" TO MOT-SOR.
*     WRITE REC-SOR.
*     MOVE 20 TO CODE-NBRE

      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
```

```
           STRING  "MOVE " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  " TO VAR" DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  PROC DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  " GO TO " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  PROC1  DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 20 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE 1 TO LONG-SOR.
           MOVE 1   TO  CODE-SOR.
           MOVE "." TO MOT-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  "WWWRET" DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  PROC DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 10 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE 1 TO LONG-SOR.
           MOVE 1   TO  CODE-SOR.
           MOVE "." TO MOT-SOR.
           PERFORM ECRIRE-MOT.


       CAS-2.
      ******
      *
      *    Correspond au point 2.2.3.1.2
      *

           MOVE SPACES TO MOT-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  "MOVE " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  " TO VAR" DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  PROC DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 20 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE "." TO MOT-SOR.
           MOVE 1   TO  CODE-SOR.
```

```
      MOVE 1 TO LONG-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "WWWRET" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  PROC DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 10 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "IF NOT " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  COND DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " GO TO " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  PROC1 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

 CAS-3.
******
*
*     Correspond au point 2.2.3.1.3
*


      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "MOVE " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " TO VAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  PROC DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
```

```
       MOVE 255 TO LONG-SOR.
       MOVE 1 TO POINTEUR.
       STRING  "MOVE " DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  NOMBRE DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  " TO NUM-TIMES" DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  NBRE-TIME DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       COMPUTE LONG-SOR = POINTEUR - 1.
       MOVE 20 TO CODE-SOR.
       PERFORM ECRIRE-MOT.



       MOVE SPACES TO MOT-SOR.
       MOVE 1 TO LONG-SOR.
       MOVE 1   TO  CODE-SOR.
       MOVE "." TO MOT-SOR.
       PERFORM ECRIRE-MOT.

       MOVE SPACES TO MOT-SOR.
       MOVE 255 TO LONG-SOR.
       MOVE 1 TO POINTEUR.
       STRING  "WWWRET" DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  PROC DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       COMPUTE LONG-SOR = POINTEUR - 1.
       MOVE 10 TO CODE-SOR.
       PERFORM ECRIRE-MOT.

       MOVE SPACES TO MOT-SOR.
       MOVE 1 TO LONG-SOR.
       MOVE 1   TO  CODE-SOR.
       MOVE "." TO MOT-SOR.
       PERFORM ECRIRE-MOT.

       MOVE SPACES TO MOT-SOR.
       MOVE 255 TO LONG-SOR.
       MOVE 1 TO POINTEUR.
       STRING  "IF (NUM-TIMES" DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  NBRE-TIME DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  " > 0) SUBTRACT 1 FROM NUM-TIMES"
                        DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  NBRE-TIME DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  " GO TO " DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  PROC1 DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       COMPUTE LONG-SOR = POINTEUR - 1.
       MOVE 20 TO CODE-SOR.
       PERFORM ECRIRE-MOT.

 CAS-4.
******
*
*    Correspond au point 2.2.3.1.4
*
```

```
           PERFORM CAS-4-A.
           PERFORM CAS-4-B
           PERFORM CAS-4-O
           PERFORM CAS-4-C.

       CAS-4-A.
      *********
           MOVE SPACES TO MOT-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  "MOVE " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  " TO VAR" DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  PROC DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 20 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  "MOVE " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  INIT1 DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  " TO " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING VAR1  DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 20 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE "." TO MOT-SOR.
           MOVE 1   TO  CODE-SOR.
           MOVE 1 TO LONG-SOR.
           PERFORM ECRIRE-MOT.

       CAS-4-B.
      *********
           MOVE SPACES TO MOT-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  "WWWRETVAR" DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  NBRE-VAR DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 10 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE "." TO MOT-SOR.
           MOVE 1   TO  CODE-SOR.
           MOVE 1 TO LONG-SOR.
           PERFORM ECRIRE-MOT.


           MOVE 255 TO LONG-SOR.
```

```
           MOVE 1 TO POINTEUR.
           STRING  "IF NOT " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  COND1 DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  " GO TO " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  PROC1 DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 20 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE 1 TO LONG-SOR.
           MOVE 1   TO  CODE-SOR.
           MOVE "." TO MOT-SOR.
           PERFORM ECRIRE-MOT.

       CAS-4-O.
      *********
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  " GO TO WWWFINVAR" DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 20 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE 1 TO LONG-SOR.
           MOVE 1   TO  CODE-SOR.
           MOVE "." TO MOT-SOR.
           PERFORM ECRIRE-MOT.

       CAS-4-C.
      *********
           MOVE SPACES TO MOT-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  "WWWRET" DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  PROC DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 10 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE 1 TO LONG-SOR.
           MOVE 1   TO  CODE-SOR.
           MOVE "." TO MOT-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  "ADD " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  INCR1 DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  " TO " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  VAR1 DELIMITED BY SIZE INTO MOT-SOR
```

```
           WITH POINTER POINTEUR.
           STRING " GO TO WWWRETVAR" DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  NBRE-VAR DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 20 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE 1 TO LONG-SOR.
           MOVE 1   TO  CODE-SOR.
           MOVE "." TO MOT-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  "WWWFINVAR" DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 10 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
           MOVE 1 TO LONG-SOR.
           MOVE 1   TO  CODE-SOR.
           MOVE "." TO MOT-SOR.
           PERFORM ECRIRE-MOT.

       CAS-5.
      ******
      *
      *    Correspond au point 2.2.3.1.4, 2eme cas
      *

           PERFORM CAS-5-D.
           PERFORM CAS-5-E.
           PERFORM CAS-5-O.
           PERFORM CAS-5-F.
           PERFORM CAS-5-G.



       CAS-5-D.
      *********
           PERFORM CAS-4-A.

           MOVE SPACES TO MOT-SOR.
           MOVE 255 TO LONG-SOR.
           MOVE 1 TO POINTEUR.
           STRING  "MOVE " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  INIT2 DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING  " TO " DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           STRING VAR2  DELIMITED BY SIZE INTO MOT-SOR
           WITH POINTER POINTEUR.
           COMPUTE LONG-SOR = POINTEUR - 1.
           MOVE 20 TO CODE-SOR.
           PERFORM ECRIRE-MOT.

           MOVE SPACES TO MOT-SOR.
```

```
      MOVE "." TO MOT-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE 1 TO LONG-SOR.
      PERFORM ECRIRE-MOT.

 CAS-5-E.
*********

      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "WWWRET2VAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NBRE-VAR DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 10 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE "." TO MOT-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE 1 TO LONG-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "IF NOT " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  COND1 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " MOVE  " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  INIT2 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " TO " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  VAR2 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  " GO TO WWWRETVAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NBRE-VAR DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

 CAS-5-O.
*********
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  " GO TO WWWFINVAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
```

```
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

 CAS-5-F.
*********
      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "WWWRETVAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NBRE-VAR DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 10 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE "." TO MOT-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE 1 TO LONG-SOR.
      PERFORM ECRIRE-MOT.


      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "IF NOT " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  COND2 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " GO TO " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  PROC1 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  " ADD " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  INCR1 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " TO " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  VAR1 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  " GO TO WWWRET2VAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
```

```
      STRING  NBRE-VAR DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

 CAS-5-G.
**********
      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "WWWRET" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  PROC DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 10 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "ADD " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  INCR2 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " TO " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  VAR2 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING " GO TO WWWRETVAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NBRE-VAR DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "WWWFINVAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NBRE-APPEL DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 10 TO CODE-SOR.
```

```
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.



 FIN-TABLE-PERF.
******************
*  ═══════════════════════════════════════════════════════════════ *
*  ═══════    I M P L E M E N T A T I O N ═══════════════════════ *
*  ═══════   DES ETAPES DE LA TRANSFORMATION  ═══════════════════ *
*  ═══════════════════    DES  IF    ═══════════════════════════════ *
*  ═══════════════ note : étapes est noté ici etatif  ═══════════ *
*  ═══════════════════════════════════════════════════════════════ *

 TRAIT-IF SECTION.
*
*╔═══════════════════╗

    MOVE 1 TO ETAT-PRESENT.
 DEBUT-TABLE-IF.
    GO TO ETAT1IF ETAT2IF ETAT3IF FIN-TABLE-IF
                                  DEPENDING ON ETAT-PRESENT.


 ETAT1IF.
*******
      ADD 1 TO NOMBRE-IF.
      MOVE NOMBRE-IF TO NOMBRE-IF-DEBUT.
      PERFORM INITIALISER-PILE.
      PERFORM AJOUTER-IF.
      PERFORM ECRIRE-MEME-MOT.
      PERFORM LIRE-SUIVANT.
      PERFORM ECRIRE-MEME-MOT.
      PERFORM CODE-1-PART.
      MOVE 2 TO ETAT-PRESENT.
      PERFORM LIRE-SUIVANT.
      GO TO DEBUT-TABLE-IF.

 ETAT2IF.
*******
      IF CODE-ENT = 1
         PERFORM ACIF1
         PERFORM LIRE-SUIVANT
      ELSE IF CODE-ENT = 31
         PERFORM ACIF2
         PERFORM LIRE-SUIVANT
      ELSE IF CODE-ENT = 30
         PERFORM ACIF3
         PERFORM LIRE-SUIVANT
      ELSE IF CODE-ENT = 22
         PERFORM ACIF4
         MOVE 2 TO ETAT-PRESENT
      ELSE PERFORM ACIF5
           PERFORM LIRE-SUIVANT.
      GO TO DEBUT-TABLE-IF.


 ETAT3IF.
      IF CODE-ENT = 1
         PERFORM ACIF1
         PERFORM LIRE-SUIVANT
```

```
      ELSE IF CODE-ENT = 31
         PERFORM ACIF6
         PERFORM LIRE-SUIVANT
      ELSE IF CODE-ENT = 30
         PERFORM ACIF3
         PERFORM LIRE-SUIVANT
      ELSE IF CODE-ENT = 22
         PERFORM ACIF4
         MOVE 3 TO ETAT-PRESENT
      ELSE PERFORM ACIF5
           PERFORM LIRE-SUIVANT.
      GO TO DEBUT-TABLE-IF.


 ACIF1.
*******
      PERFORM CODE-3-PART.
      MOVE 4 TO ETAT-PRESENT.

 ACIF2.
*******
      PERFORM AJOUTER-ELSE.
      PERFORM CODE-2-PART.
      MOVE 3 TO ETAT-PRESENT.

 ACIF3.
*******
      ADD 1 TO NOMBRE-IF.
      PERFORM AJOUTER-IF.
      PERFORM ECRIRE-MEME-MOT.
      PERFORM LIRE-SUIVANT.
      PERFORM ECRIRE-MEME-MOT.
      PERFORM CODE-1-PART.
      MOVE 2 TO ETAT-PRESENT.

 ACIF4.
*******
      PERFORM TRAIT-PERF.

 ACIF5.
*******
      PERFORM ECRIRE-MEME-MOT.

 ACIF6.
*******
      PERFORM AJOUTER-ELSE.
      PERFORM CODE-2-PART.


*  ================================================================  *
*  ==========    ECRITURE DES MOTS COBOL DANS ==================    *
*  ==========  LE FICHIER DE SORTIE            ==================    *
*  ================================================================  *


 CODE-1-PART.
*************

      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "GO TO WWWIFTHEN" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NOMBRE-IF DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " ELSE GO TO WWWIFELSE" DELIMITED BY SIZE
```

```
           INTO MOT-SOR WITH POINTER POINTEUR.
       STRING  NOMBRE-IF DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       COMPUTE LONG-SOR = POINTEUR - 1.
       MOVE 20 TO CODE-SOR.
       PERFORM ECRIRE-MOT.

       MOVE SPACES TO MOT-SOR.
       MOVE 1 TO LONG-SOR.
       MOVE 1   TO  CODE-SOR.
       MOVE "." TO MOT-SOR.
       PERFORM ECRIRE-MOT.

       MOVE SPACES TO MOT-SOR.
       MOVE 255 TO LONG-SOR.
       MOVE 1 TO POINTEUR.
       STRING  "WWWIFTHEN" DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  NOMBRE-IF DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       COMPUTE LONG-SOR = POINTEUR - 1.
       MOVE 10 TO CODE-SOR.
       PERFORM ECRIRE-MOT.

       MOVE SPACES TO MOT-SOR.
       MOVE 1 TO LONG-SOR.
       MOVE 1   TO  CODE-SOR.
       MOVE "." TO MOT-SOR.
       PERFORM ECRIRE-MOT.

 CODE-2-PART.
**************

       MOVE SPACES TO MOT-SOR.
       MOVE 255 TO LONG-SOR.
       MOVE 1 TO POINTEUR.
       STRING  "GO TO WWWFINIF" DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  NOMBRE-IF-DEBUT DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       COMPUTE LONG-SOR = POINTEUR - 1.
       MOVE 20 TO CODE-SOR.
       PERFORM ECRIRE-MOT.

       MOVE SPACES TO MOT-SOR.
       MOVE 1 TO LONG-SOR.
       MOVE 1   TO  CODE-SOR.
       MOVE "." TO MOT-SOR.
       PERFORM ECRIRE-MOT.

       PERFORM CODE-2-2.


 CODE-2-2.
***********
       MOVE SPACES TO MOT-SOR.
       MOVE 255 TO LONG-SOR.
       MOVE 1 TO POINTEUR.
       STRING  "WWWIFELSE" DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       STRING  IF-EN-COURS DELIMITED BY SIZE INTO MOT-SOR
       WITH POINTER POINTEUR.
       COMPUTE LONG-SOR = POINTEUR - 1.
       MOVE 10 TO CODE-SOR.
       PERFORM ECRIRE-MOT.
```

```
      MOVE SPACES TO MOT-SOR.
      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.


 ECRIRE-THEN-NON-APPARIE.
**************************

      MOVE 0 TO TROUVE-NUMIF.
      ADD 1 TO POINTEUR-PILE-2.

      PERFORM CHERCHER-IF-NON-APPARIE.

      IF TROUVE-NUMIF = 1
         PERFORM CODE-2-2.



 CODE-3-PART.
*************

      MOVE SPACES TO MOT-SOR.
      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 0 TO POINTEUR-PILE-2.
      PERFORM ECRIRE-THEN-NON-APPARIE VARYING IF-EN-COURS
      FROM NOMBRE-IF-DEBUT BY 1 UNTIL (IF-EN-COURS > NOMBRE-IF).

      MOVE SPACES TO MOT-SOR.
      MOVE 255 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "WWWFINIF" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NOMBRE-IF-DEBUT DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 10 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE SPACES TO MOT-SOR.
      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

*    ================================================================ *
* ================  P R I M I T I V E S  ================================ *
*    ================================================================ *

 INITIALISER-PILE.
******************
      MOVE 0 TO POINTEUR-PILE.

 AJOUTER-IF.
*************
      ADD 1 TO POINTEUR-PILE.
      MOVE NOMBRE-IF TO NUM-IF (POINTEUR-PILE).
      MOVE 0 TO IF-APPARIE (POINTEUR-PILE).

 AJOUTER-ELSE.
```

```
***************

      MOVE 0 TO TROUVE-NUMIF.
      PERFORM CHERCHER-IF-NON-APPARIE
              VARYING POINTEUR-PILE-2 FROM POINTEUR-PILE BY -1
              UNTIL ( TROUVE-NUMIF = 1 ).
      ADD 1 TO POINTEUR-PILE-2.
      MOVE NUM-IF (POINTEUR-PILE-2) TO IF-EN-COURS.


 CHERCHER-IF-NON-APPARIE.
**************************
      IF IF-APPARIE (POINTEUR-PILE-2) = 0
         MOVE 1 TO IF-APPARIE (POINTEUR-PILE-2)
         MOVE 1 TO TROUVE-NUMIF.

*  ═══════════════════════════════════════════════════════════════════  *
*  ═══════════════════════════════════════════════════════════════════  *
*  ═══════════════════════════════════════════════════════════════════  *

 FIN-TABLE-IF.
***************

*
*    ╔═══════════════════════════════════════════════════════════════╗
*    ║     TRANSFORMATION DES PERFORM ET DES IF                      ║
*    ║          FIN DE LA PREMIERE PASSE                             ║
*    ║                                                               ║
*    ╚═══════════════════════════════════════════════════════════════╝
```

```
*
*
*       S U P P R E S S I O N   D E S   P E R F O R M
*                    DEUXIEME PASSE
*
*
*
 IDENTIFICATION DIVISION.
**************************
 PROGRAM-ID.  PER-2-PASS.


 ENVIRONMENT DIVISION.
***********************
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

 SELECT FICHIER-ENTREE  ASSIGN TO FICHIER-SOURCE
        ORGANIZATION IS SEQUENTIAL.

 SELECT FICHIER-SORTIE ASSIGN TO FICHIER-RESULTAT
        ORGANIZATION IS SEQUENTIAL.

 SELECT TABLE-TITRE ASSIGN TO "TABLE.PER"
        ORGANIZATION IS INDEXED
        ACCESS MODE IS DYNAMIC
        RECORD KEY IS NOM-TITRE.


 DATA DIVISION.
****************

 FILE SECTION.
***************
 FD FICHIER-ENTREE LABEL RECORD IS STANDARD.
 01 REC-ENTREE.
    02 CODE-ENT       PIC 99.
    02 LONG-ENT       PIC 999.
    02 MOT-ENT.
       03 CAR-ENT PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-ENT.

 FD FICHIER-SORTIE LABEL RECORD IS STANDARD.
 01 REC-SORTIE.
    02 CODE-SOR       PIC 99.
    02 LONG-SOR       PIC 999.
    02 MOT-SOR.
       03 CAR-SOR PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-SOR.

 FD TABLE-TITRE LABEL RECORD IS STANDARD.
 01 REC-TITRE.
    02 NOM-TITRE PIC X(30).
    02 NBRE-APPEL PIC 99.


 WORKING-STORAGE SECTION.
**************************

 01 FICHIER-SOURCE    PIC X(12).
 01 FICHIER-RESULTAT  PIC X(12).


 01 DEBUT-A   PIC 99 VALUE 8.
```

```
 01 DEBUT-B   PIC 99 VALUE 12.
 01 FIN-B     PIC 99 VALUE 72.
 01 DEBUT-C   PIC 99 VALUE 7.

 01 I                  PIC 99.

 01 COMPT-PARENTHESE   PIC 99.
 01 NBRE-ESPACE        PIC 99.
 01 FIN-LIGNE          PIC 9.
 01 ETAT-PRESENT   PIC 99.
 01 CAS                PIC 99.
 01 POINTEUR           PIC 999.

 01 TROUVE PIC 9.
 01 TROUVE-TITRE PIC 9.

 01 FIN-TITRE      PIC 9.
 01 FIN-FICHIER    PIC 9.

 01 TITRE-COURANT        PIC X(30).
 01 NBRE-APPEL-TITRE     PIC 99.
 01 SECTION-COURANTE     PIC X(30).
 01 NBRE-APPEL-SECTION   PIC 99.

 01 TITRE-EXAMINE PIC X(30).
 01 TITRE-RED REDEFINES TITRE-EXAMINE.
    02 PREFIXE PIC XXX.
    02 RESTE-TITRE PIC X(27).


 01 MOT-GARDE.
    02 CODE-GARDE    PIC 99.
    02 LONG-GARDE    PIC 999.
    02 MOT-GAR.
       03 CAR-GAR PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-GARDE.


 01 NBRE-TIME PIC 99.
 01 NBRE-VAR  PIC 99.
 01 NOMBRE-IF PIC 99.
 01 IF-EN-COURS PIC 99.


 PROCEDURE DIVISION.
**********************
 PRINCIPAL SECTION.
**********************
*  ================================================================  *
*  ============== S A I S I E  D E S  P A R A M E T R E S ==========  *
*  ================================================================  *

     DISPLAY "entrez le fichier source (.P1)" UPON CONSOLE.
     ACCEPT FICHIER-SOURCE FROM CONSOLE.
     MOVE FICHIER-SOURCE TO FICHIER-RESULTAT.
     INSPECT FICHIER-SOURCE REPLACING FIRST "    " BY ".P1".
     INSPECT FICHIER-RESULTAT REPLACING FIRST "    " BY ".MOT".

     DISPLAY "le fichier source est " FICHIER-SOURCE
             "et le fichier resultat est " FICHIER-RESULTAT
                                          UPON CONSOLE.

*  ================================================================  *
*  ============== D E B U T  D U  P R O G R A M M E ================  *
*  ================================================================  *
```

```
      OPEN INPUT FICHIER-ENTREE.
      OPEN OUTPUT FICHIER-SORTIE.
      OPEN INPUT TABLE-TITRE.

      PERFORM LIRE-SUIVANT.
      PERFORM RECOPIER-MOT UNTIL MOT-ENT = "WORKING-STORAGE".
      PERFORM RECOPIER-MOT.
      PERFORM RECOPIER-MOT.
      PERFORM ECRIRE-MEME-MOT.

*
*   nous sommes au début de la WORKING-STORAGE SECTION.
*   nous allons déclarer les variables qui dépendent des
*   titres de procedure  (trait-titre)
*   et les variables nombre-perform
*

      MOVE 0 TO FIN-TITRE.
      PERFORM LIRE-TITRE.
      PERFORM TRAIT-TITRE UNTIL FIN-TITRE = 1.
      PERFORM TRAIT-NBRE-TIME.

      PERFORM LIRE-SUIVANT.
      PERFORM RECOPIER-MOT UNTIL MOT-ENT = "DIVISION".
      PERFORM RECOPIER-MOT.

      PERFORM INIT-TITRE.
      PERFORM INIT-SECTION.
      PERFORM TRAITEMENT UNTIL FIN-FICHIER = 1.

      IF TITRE-COURANT NOT = SPACES
         PERFORM TRAIT-RETOUR.
      IF SECTION-COURANTE NOT = SPACES
         PERFORM TRAIT-RETOUR-SECTION.


      CLOSE FICHIER-ENTREE.
      CLOSE FICHIER-SORTIE.
      CLOSE TABLE-TITRE.
      STOP RUN.
*****************************************************************
*****************************************************************

*    ========================================================== *
* ========== P R I M I T I V E S ========================== *
*  ========================================================== *

 LIRE-SUIVANT.
***************
      READ FICHIER-ENTREE AT END MOVE 1 TO FIN-FICHIER.

 ECRIRE-MOT.
*************
      WRITE REC-SORTIE.

 ECRIRE-MEME-MOT.
*************
      WRITE REC-SORTIE FROM REC-ENTREE.


 CHERCHER-TITRE.
*
*  cherche si proc se trouve dans le fichier table-titre.
*  si oui, donne sa position et met trouve-titre = 1
*  si non, met trouve-titre = 0
*
```

```
      MOVE 1 TO TROUVE-TITRE.
      MOVE MOT-ENT TO NOM-TITRE.
      READ TABLE-TITRE RECORD
           KEY IS NOM-TITRE
           INVALID KEY MOVE O TO TROUVE-TITRE.

*  ================================================================  *
 GARDER-MOT-LU.
***************
      MOVE LONG-ENT TO LONG-GARDE.
      MOVE REC-ENTREE TO MOT-GARDE.

 ECRIRE-MOT-GARDE.
******************
     WRITE REC-SORTIE FROM MOT-GARDE.

 RECOPIER-MOT.
**************
      PERFORM ECRIRE-MEME-MOT.
      PERFORM LIRE-SUIVANT.


 LIRE-TITRE.
************  lire dans le fichier table-titre l'élément suivant.
      READ TABLE-TITRE NEXT RECORD AT END MOVE 1 TO FIN-TITRE.


 TRAIT-TITRE.
*************
*
*   va déclarer (en WSS) les titres de procedures
*   qui se trouvent dans la table des titres de
*   procedure.
*

      MOVE 50 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING  "01 VAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  NOM-TITRE DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " PIC 99" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 10 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

      PERFORM LIRE-TITRE.


 TRAIT-NBRE-TIME.
******************
      MOVE "NOMBRE-TIME" TO NOM-TITRE.
      READ TABLE-TITRE RECORD INVALID KEY MOVE O TO NBRE-APPEL.
      PERFORM TRAIT-1-TIME VARYING I FROM 1 BY 1
                           UNTIL (I > NBRE-APPEL).

 TRAIT-1-TIME.
***************
      MOVE 50 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
```

```
      STRING  "01 NUM-TIMES" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  I DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  " PIC 99" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 10 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.

 INIT-TITRE.
************
      MOVE SPACES TO TITRE-COURANT.
      MOVE 0 TO NBRE-APPEL-TITRE.

 INIT-SECTION.
**************
      MOVE SPACES TO SECTION-COURANTE.
      MOVE 0 TO NBRE-APPEL-SECTION.

 TRAITEMENT.
************
      MOVE MOT-ENT TO TITRE-EXAMINE.

      IF (CODE-ENT = 10 AND  PREFIXE NOT = "WWW")
         PERFORM TRAIT-TITRE-ORIGINAL.

      PERFORM RECOPIER-MOT.


 TRAIT-TITRE-ORIGINAL.
**********************
      IF TITRE-COURANT NOT = SPACES
         PERFORM TRAIT-RETOUR.

      PERFORM CHERCHER-TITRE.
      IF TROUVE-TITRE = 1
         MOVE MOT-ENT TO TITRE-COURANT
         MOVE NBRE-APPEL TO NBRE-APPEL-TITRE
      ELSE
         PERFORM INIT-TITRE.

      PERFORM GARDER-MOT-LU.
      PERFORM LIRE-SUIVANT.
      IF MOT-ENT = "SECTION"
         PERFORM TRAIT-SECTION.
      PERFORM ECRIRE-MOT-GARDE.



 TRAIT-SECTION.
***************
         IF SECTION-COURANTE NOT = SPACES
            PERFORM TRAIT-RETOUR-SECTION.

         PERFORM CHERCHER-TITRE
         IF TROUVE-TITRE = 1
            MOVE MOT-ENT TO SECTION-COURANTE
            MOVE NBRE-APPEL TO NBRE-APPEL-SECTION
         ELSE
```

```
           PERFORM INIT-SECTION.


 TRAIT-RETOUR.
***************

      MOVE 6 TO LONG-SOR.
      MOVE 20  TO  CODE-SOR.
      MOVE "GO TO " TO MOT-SOR.
      PERFORM ECRIRE-MOT.

      PERFORM ECRIRE-RET-PROC-I VARYING I FROM 1 BY 1
                                UNTIL I > NBRE-APPEL-TITRE.
      IF NBRE-APPEL-TITRE = 1
         MOVE 1 TO I
         PERFORM ECRIRE-RET-PROC-I.

      MOVE 1 TO POINTEUR.
      MOVE 50 TO LONG-SOR.
      STRING  " DEPENDING ON VAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  TITRE-COURANT DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.


 ECRIRE-RET-PROC-I.
********************

      MOVE 50 TO LONG-SOR.
      MOVE SPACES TO MOT-SOR.
      MOVE 1 TO POINTEUR.
      STRING  " WWWRET" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  TITRE-COURANT DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.

      INSPECT MOT-SOR REPLACING FIRST "  " BY I.
      COMPUTE LONG-SOR = POINTEUR + 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

 TRAIT-RETOUR-SECTION.
***********************

      MOVE 6 TO LONG-SOR.
      MOVE 20  TO  CODE-SOR.
      MOVE "GO TO " TO MOT-SOR.
      PERFORM ECRIRE-MOT.

      PERFORM ECRIRE-RET-SECT-I VARYING I FROM 1 BY 1
                                UNTIL I > NBRE-APPEL-SECTION.
      IF NBRE-APPEL-SECTION = 1
         MOVE 1 TO I
         PERFORM ECRIRE-RET-SECT-I.

      MOVE 50 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
```

```
      STRING  " DEPENDING ON VAR" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  SECTION-COURANTE DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      COMPUTE LONG-SOR = POINTEUR - 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.

      MOVE 1 TO LONG-SOR.
      MOVE 1   TO  CODE-SOR.
      MOVE "." TO MOT-SOR.
      PERFORM ECRIRE-MOT.


 ECRIRE-RET-SECT-I.
*******************

      MOVE 50 TO LONG-SOR.
      MOVE SPACES TO MOT-SOR.
      MOVE 1 TO POINTEUR.
      STRING  " WWWRET" DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING  SECTION-COURANTE DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      INSPECT MOT-SOR REPLACING FIRST "  " BY I.
      COMPUTE LONG-SOR = POINTEUR + 1.
      MOVE 20 TO CODE-SOR.
      PERFORM ECRIRE-MOT.
```

```
*
*
*       INVERSION DE P1 PAR RAPPORT A P2
*       MODIFIE LE PROGRAMME P1 ET MET LE RESULTAT DANS P1-1
*
*
 IDENTIFICATION DIVISION.
**************************
 PROGRAM-ID.  INVP1.

 ENVIRONMENT DIVISION.
***********************
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
***************
 SELECT FICHTEMP  ASSIGN TO "TEMPO.XXX"
        ORGANIZATION IS SEQUENTIAL.

 SELECT FICHIER-ENTREE  ASSIGN TO FICHIER-SOURCE
        ORGANIZATION IS SEQUENTIAL.

 SELECT FICHIER-SORTIE ASSIGN TO "SOR.XXX"
        ORGANIZATION IS SEQUENTIAL.

 SELECT FICH-INVERSE  ASSIGN TO FICHIER-RESULTAT
        ORGANIZATION IS SEQUENTIAL.


 DATA DIVISION.
*****************

 FILE SECTION.
***************
 FD FICHTEMP LABEL RECORD IS STANDARD.
 01 REC-TEM.
    02 CODE-TEM       PIC 99.
    02 LONG-TEM       PIC 999.
    02 MOT-TEM.
       03 CAR-TEM PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-TEM.

 FD FICHIER-ENTREE LABEL RECORD IS STANDARD.
 01 REC-ENT.
    02 CODE-ENT       PIC 99.
    02 LONG-ENT       PIC 999.
    02 MOT-ENT.
       03 CAR-ENT PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-ENT.

 FD FICH-INVERSE LABEL RECORD IS STANDARD.
 01 REC-INV.
    02 CODE-INV       PIC 99.
    02 LONG-INV       PIC 999.
    02 MOT-INV.
       03 CAR-INV PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-INV.

 FD FICHIER-SORTIE LABEL RECORD IS STANDARD.
 01 REC-SOR.
    02 CODE-SOR       PIC 99.
    02 LONG-SOR       PIC 999.
    02 MOT-SOR.
       03 CAR-SOR PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-SOR.
```

```
 WORKING-STORAGE SECTION.
************************

 01 MOT-GARDE.
    02 CODE-GARDE     PIC 99.
    02 LONG-GARDE     PIC 999.
    02 MOT-GAR.
       03 CAR-GAR PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-GARDE.

 01 FICHIER-SOURCE        PIC X(12).
 01 FICHIER-RESULTAT      PIC X(12).
 01 NOM-RESULTAT          PIC X(12).

 01 P1                PIC X(30).
 01 P2                PIC X(30).
 01 FICH-C            PIC X(30).
 01 REC-C             PIC X(30).
 01 REC1              PIC X(30).

 01 POINTEUR          PIC 9(3).
 01 I                 PIC 99.

 01 NBRE-WRITE      PIC 99.

 01 ETAT-PRESENT    PIC 99.

 01 TROUVE-FICHC    PIC 9.
 01 TROUVE-AT-END   PIC 9.

 01 FIN-FICHIER     PIC 9.
 01 FIN-SOR         PIC 9.
 01 FIN-FICHTEMP    PIC 9.


 PROCEDURE DIVISION.
*********************

*_____________________________________________________________________
*_____________ SAISIE DES PARAMETRES P1, P2, FICHC _____________________
*_____________________________________________________________________

      DISPLAY "INVERSION DE P1 % P2, MODIF P1" UPON CONSOLE.
      DISPLAY "entrez le fichier P1 (.MOT)" UPON CONSOLE.
      ACCEPT FICHIER-SOURCE FROM CONSOLE.
      MOVE FICHIER-SOURCE TO NOM-RESULTAT.
      INSPECT FICHIER-SOURCE REPLACING FIRST "    " BY ".MOT".
      INSPECT NOM-RESULTAT REPLACING FIRST "  " BY "-1".
      MOVE  NOM-RESULTAT TO FICHIER-RESULTAT.
      INSPECT FICHIER-RESULTAT REPLACING FIRST "    " BY ".MOT".

      DISPLAY "le fichier source P1 est " FICHIER-SOURCE
              "et le fichier resultat P1-1 est " FICHIER-RESULTAT
                                                 UPON CONSOLE.


      DISPLAY "entrez le nom du fichier intermédiaire "
                                 UPON CONSOLE.
      ACCEPT FICH-C FROM CONSOLE.

*_____________________________________________________________________
*_____________________________________________________________________
*_____________ TRAITEMENT DE P1, 1 ERE PASSE ____________________
```

```
*
      OPEN INPUT FICHIER-ENTREE
      OPEN OUTPUT FICHIER-SORTIE
      OPEN OUTPUT FICHTEMP.

*
*     modification du nom du programme
*
      PERFORM  LIRE-SUIVANT.
      PERFORM ECRIRE-LIRE UNTIL (MOT-ENT = "PROGRAM-ID").
      PERFORM ECRIRE-LIRE.
      PERFORM ECRIRE-LIRE.
* on a lu le nom du programme et on va le remplacer par le nouveau
      MOVE 10 TO LONG-ENT
      MOVE 1 TO POINTEUR.
      STRING NOM-RESULTAT DELIMITED BY SIZE INTO MOT-ENT
      WITH POINTER POINTEUR.
      MOVE POINTEUR TO LONG-ENT.
      MOVE 20 TO CODE-ENT.

*
*     suppression du SELECT FICH-C
*
      PERFORM ECRIRE-LIRE UNTIL (MOT-ENT = "FILE-CONTROL").
      MOVE 0 TO TROUVE-FICHC.
      PERFORM TRAIT-SELECT UNTIL (TROUVE-FICHC = 1).

*
*     suppression de FD FICHC et
*     sauvetage de la declaretion de REC
*
      PERFORM ECRIRE-LIRE UNTIL (MOT-ENT = "FILE").
      PERFORM ECRIRE-LIRE.
      MOVE 0 TO TROUVE-FICHC.
      PERFORM TRAIT-FD UNTIL (TROUVE-FICHC = 1).
      PERFORM ECRIRE-LIRE.

*
*     ajout de la declaration de QS  en WSS
*
* ici, on a ecrit WORKING-STORAGE SECTION. et lu le mot suivant
      PERFORM ECRIRE-01QSPIC99VALUE1.
      PERFORM ECRIRE-POINT.

*
*     ajout de la declaration de REC et EOF
*     en LINKAGE SECTION  et creation de la
*     LINKAGE SECTION si elle n'existait pas
*
      PERFORM ECRIRE-LIRE UNTIL (((MOT-ENT = "PROCEDURE")
                                  OR (MOT-ENT = "COMMUNICATION"))
                                  OR (MOT-ENT = "LINKAGE")).

      IF (MOT-ENT  = "LINKAGE")
         PERFORM ECRIRE-LIRE
         PERFORM ECRIRE-LIRE
         PERFORM ECRIRE-LIRE
      ELSE
```

```
           PERFORM ECRIRE-LINKAGESECTION
           PERFORM ECRIRE-POINT.
* ici, nous sommes au debut de la linkage section

       CLOSE FICHTEMP
       OPEN INPUT FICHTEMP.
       PERFORM RECOPIER-FICHTEMP.
       PERFORM ECRIRE-01EOFPIC9
       PERFORM ECRIRE-POINT.

*
*  modifications de la PROCEDURE DIVISION
*
       IF (MOT-ENT NOT = "PROCEDURE")
          PERFORM ECRIRE-LIRE UNTIL (MOT-ENT = "PROCEDURE").
       PERFORM ECRIRE-LIRE.
       MOVE 1 TO NBRE-WRITE.
       PERFORM TRAIT-PROC-DIV UNTIL (FIN-FICHIER = 1).

       CLOSE FICHIER-ENTREE
       CLOSE FICHIER-SORTIE
       CLOSE FICHTEMP.


*
*
*  TRAITEMENT DE P1, 2 EME PASSE
*

       OPEN INPUT FICHIER-SORTIE
       OPEN OUTPUT FICH-INVERSE

       MOVE 0 TO FIN-SOR.
       PERFORM LIRE-SOR.
       PERFORM ECR-LIRE UNTIL (MOT-SOR = "PROCEDURE").
       PERFORM ECR-LIRE
       PERFORM ECR-LIRE

* nous avons ecrit PROCEDURE DIVISION
       PERFORM ECR-USINGEOF
       PERFORM ECR-REC


       PERFORM ECR-LIRE.
* Nous avons écrit le point suivant procedure division.

       PERFORM ECR-GOTO.
       PERFORM ECR-QI VARYING I FROM 1 BY 1 UNTIL (I > NBRE-WRITE)
       PERFORM ECR-DEPENDINGONQS
       PERFORM ECR-POINT
       PERFORM ECR-Q1.
       PERFORM ECR-POINT.


       PERFORM ECR-LIRE UNTIL (FIN-SOR = 1).



       CLOSE FICHIER-SORTIE
       CLOSE FICH-INVERSE

       STOP RUN.

*                                                                *
*
*  procedures de la 1 ERE PASSE
```

```
*  ═══════════════════════════════════════════════════════════════════
*  ═══════════════════════════════════════════════════════════════════ *

 TRAIT-SELECT.
**************
      PERFORM ECRIRE-LIRE UNTIL (CODE-ENT = 40).
      PERFORM LIRE-SUIVANT.
      IF (MOT-ENT = FICH-C)
         PERFORM LIRE-SUIVANT UNTIL (CODE-ENT = 1)
         PERFORM LIRE-SUIVANT
         MOVE 1 TO TROUVE-FICHC
      ELSE
         PERFORM ECRIRE-SELECT.

*  ═══════════════════════════════════════════════════════════════════ *

 TRAIT-FD.
***********
      PERFORM ECRIRE-LIRE UNTIL (CODE-ENT = 41).                        SELECT
      PERFORM LIRE-SUIVANT.
      IF (MOT-ENT = FICH-C)
         PERFORM LIRE-SUIVANT UNTIL (CODE-ENT = 1)                      POINT
         MOVE 1 TO TROUVE-FICHC
*             ici, il faut identifier REC

         PERFORM LIRE-SUIVANT
         PERFORM ECRIRE-LIRE-FICHTEMP UNTIL (CODE-ENT = 41           SELECT
                                  OR MOT-ENT = "WORKING-STORAGE")
         IF (MOT-ENT = "WORKING-STORAGE")
            PERFORM ECRIRE-LIRE
            PERFORM ECRIRE-LIRE
         ELSE
            PERFORM ECRIRE-LIRE UNTIL (MOT-ENT =
                                       "WORKING-STORAGE")
            PERFORM ECRIRE-LIRE
            PERFORM ECRIRE-LIRE

      ELSE
         PERFORM ECRIRE-SELECT.


*  ═══════════════════════════════════════════════════════════════════ *

 TRAIT-PROC-DIV.
*****************
*
*  ╔═══════════════════════════════════════╗
*  ║ traitement de la PROCEDURE DIVISION   ║
*  ╚═══════════════════════════════════════╝
*


      IF (MOT-ENT = "OPEN" )
         PERFORM TRAIT-OPEN
      ELSE
      IF (MOT-ENT = "WRITE" )
         PERFORM TRAIT-WRITE
      ELSE
      IF (MOT-ENT = "CLOSE" )
         PERFORM TRAIT-CLOSE
      ELSE
      IF (MOT-ENT = "STOP" )
         PERFORM TRAIT-STOP-RUN.
      PERFORM ECRIRE-MEME-MOT.
      PERFORM LIRE-SUIVANT.
```

```
*  ================================================================ *

 TRAIT-OPEN.
************
*
* si OPEN OUTPUT FICH-C, supprimer cette instruction
* sinon, la recopier
*

      PERFORM GARDER-FICHTEMP
      PERFORM LIRE-SUIVANT
      IF MOT-ENT = "OUTPUT"
         PERFORM GARDER-FICHTEMP
         PERFORM LIRE-SUIVANT
         IF MOT-ENT = FICH-C
            PERFORM VIDER-FICHTEMP
            PERFORM LIRE-SUIVANT
            IF MOT-ENT = "."
               PERFORM LIRE-SUIVANT
            ELSE
               NEXT SENTENCE
         ELSE
            PERFORM RECOPIER-FICHTEMP
      ELSE
         PERFORM RECOPIER-FICHTEMP.

*  ================================================================ *

 TRAIT-WRITE.
************
*
*  si WRITE REC-C, appel à TRAIT-WRITE-REC-C
*  sinon, recopier les instructions
*
      PERFORM GARDER-FICHTEMP
      PERFORM LIRE-SUIVANT
      IF MOT-ENT = REC-C
         PERFORM VIDER-FICHTEMP
         PERFORM TRAIT-WRITE-REC-C
      ELSE
         PERFORM RECOPIER-FICHTEMP.


*  ================================================================ *

 TRAIT-CLOSE.
************
*
*  si CLOSE FICH-C, supprimer
*  sinon, recopier les instructions
*
      PERFORM GARDER-FICHTEMP
      PERFORM LIRE-SUIVANT
      IF MOT-ENT = FICH-C
         PERFORM VIDER-FICHTEMP
         PERFORM LIRE-SUIVANT
      ELSE
         PERFORM RECOPIER-FICHTEMP.


*  ================================================================ *

 TRAIT-STOP-RUN.
****************
*
* Le STOP a été lu : il faut lire le RUN et remplacer
```

```
* le tout par MOVE 1 TO EOF
*          DERNIERE-SORTIE.
*             EXIT PROGRAM.
*          FIN-DE-PROGRAMME.


      PERFORM LIRE-SUIVANT.
      PERFORM LIRE-SUIVANT.

      PERFORM ECRIRE-MOVE1TOEOF.
      PERFORM ECRIRE-POINT.
      PERFORM ECRIRE-DERNIERESORTIE.
      PERFORM ECRIRE-POINT.
      PERFORM ECRIRE-EXITPROGRAM.
      PERFORM ECRIRE-POINT.
      PERFORM ECRIRE-FINPROGRAM.
      PERFORM ECRIRE-POINT.

*  pour éviter d'écrire 2 points successivement,
     IF CODE-ENT = 1
        PERFORM LIRE-SUIVANT.



* ═══════════════════════════════════════════════════════════════ *

 TRAIT-WRITE-REC-C.
******************
* EN ENTRANT ICI, ON A LU : WRITE REC-C

      ADD 1 TO NBRE-WRITE.
      PERFORM LIRE-SUIVANT.

* tester si on a "FROM REC1"
      MOVE SPACES TO REC1.
      IF MOT-ENT = "FROM"
         PERFORM LIRE-SUIVANT
         MOVE MOT-ENT TO REC1
         PERFORM LIRE-SUIVANT.

      PERFORM ECRIRE-MOVE.
      PERFORM ECRIRE-NBRE-WRITE.
      PERFORM ECRIRE-TOQSMOVEOTOEOF.

      IF REC1 NOT = SPACES
         PERFORM ECRIRE-MOVE
         PERFORM ECRIRE-REC1
         PERFORM ECRIRE-TO
         PERFORM ECRIRE-REC-C.
      PERFORM ECRIRE-POINT.

      PERFORM ECRIRE-SORTIENBRE-WRITE.
      PERFORM ECRIRE-POINT.
      PERFORM ECRIRE-EXITPROGRAM.
      PERFORM ECRIRE-POINT.
      PERFORM ECRIRE-QNBRE-WRITE.
      PERFORM ECRIRE-POINT.

*  pour éviter d'écrire 2 points successivement,
     IF CODE-ENT = 1
        PERFORM LIRE-SUIVANT.


*  ═══════════════════════════════════════════════════════════════ *
* ═══════════════  P R I M I T I V E S  POUR LA 1ERE PASSE  ═══ *
* ═══════════════════════════════════════════════════════════════ *
```

```
 LIRE-SUIVANT.
***************
      READ FICHIER-ENTREE AT END MOVE 1 TO FIN-FICHIER.

 ECRIRE-MOT.
*************
      WRITE REC-SOR.

 ECRIRE-MEME-MOT.
*************
      WRITE REC-SOR FROM REC-ENT.


 ECRIRE-LIRE.
***************
      PERFORM ECRIRE-MEME-MOT.
      PERFORM LIRE-SUIVANT.


 GARDER-FICHTEMP.
******************
* VA ECRIRE LE MOT LU DANS FICHTEMP

      WRITE REC-TEM FROM REC-ENT.


 ECRIRE-LIRE-FICHTEMP.
***********************
      PERFORM GARDER-FICHTEMP
      PERFORM LIRE-SUIVANT.

 RECOPIER-FICHTEMP.
********************
* va ecrire le contenu de fichtemp dans le fichier de sortie
* puis réinitialiser fichtemp, en faisant CLOSE FICHTEMP
*  et OPEN OUTPUT FICHTEMP

      CLOSE FICHTEMP.
      OPEN INPUT FICHTEMP.
      MOVE 0 TO FIN-FICHTEMP.
      PERFORM ECRIRE-FICHTEMP UNTIL FIN-FICHTEMP = 1.
      CLOSE FICHTEMP
      OPEN OUTPUT FICHTEMP.


 ECRIRE-FICHTEMP.
******************
      READ FICHTEMP
            AT END MOVE 1 TO FIN-FICHTEMP.
      IF FIN-FICHTEMP = 0
         WRITE REC-SOR FROM REC-TEM.

 VIDER-FICHTEMP.
*****************
      CLOSE FICHTEMP.
      OPEN OUTPUT FICHTEMP.

*
*
*
*     E C R I T U R E   D U   C O D E   POUR LA 1ERE PASSE
*
*
*

 ECRIRE-SELECT.
```

```
***************
     MOVE 6 TO LONG-SOR
     MOVE "SELECT" TO MOT-SOR.
     MOVE 10 TO CODE-SOR.
     WRITE REC-SOR.


 ECRIRE-REC1.
**************
     MOVE 250 TO LONG-SOR.
     MOVE 1 TO POINTEUR.
     STRING REC1 DELIMITED BY SIZE INTO MOT-SOR
     WITH POINTER POINTEUR.
     MOVE POINTEUR TO LONG-SOR.
     MOVE 20 TO CODE-SOR.
     WRITE REC-SOR.

 ECRIRE-REC-C.
*************
     MOVE 250 TO LONG-SOR.
     MOVE 1 TO POINTEUR.
     STRING REC-C DELIMITED BY SIZE INTO MOT-SOR
     WITH POINTER POINTEUR.
     MOVE POINTEUR TO LONG-SOR.
     MOVE 20 TO CODE-SOR.
     WRITE REC-SOR.



 ECRIRE-01EOFPIC9.
*******************
     MOVE 13 TO LONG-SOR
     MOVE "01 EOF PIC 9 " TO MOT-SOR.
     MOVE 20 TO CODE-SOR.
     WRITE REC-SOR.

 ECRIRE-01QSPIC99VALUE1.
************************
     MOVE 21 TO LONG-SOR
     MOVE "01 QS PIC 99 VALUE 1 " TO MOT-SOR.
     MOVE 10 TO CODE-SOR.
     WRITE REC-SOR.

 ECRIRE-LINKAGESECTION.
************************
     MOVE 15 TO LONG-SOR
     MOVE "LINKAGE SECTION" TO MOT-SOR.
     MOVE 10 TO CODE-SOR.
     WRITE REC-SOR.

 ECRIRE-POINT.
**************
     MOVE 1 TO LONG-SOR
     MOVE "." TO MOT-SOR.
     MOVE 1 TO CODE-SOR.
     WRITE REC-SOR.

 ECRIRE-MOVE.
*************
     MOVE 4  TO LONG-SOR
     MOVE "MOVE" TO MOT-SOR.
     MOVE 20 TO CODE-SOR.
     WRITE REC-SOR.


 ECRIRE-NBRE-WRITE.
```

```
********************
     MOVE 250 TO LONG-SOR.
     MOVE 1 TO POINTEUR.
     STRING NBRE-WRITE DELIMITED BY SIZE INTO MOT-SOR
     WITH POINTER POINTEUR.
     MOVE 2 TO LONG-SOR.
     MOVE 20 TO CODE-SOR.
     WRITE REC-SOR.


 ECRIRE-TOQSMOVEOTOEOF.
************************
     MOVE 19 TO LONG-SOR
     MOVE "TO QS MOVE O TO EOF" TO MOT-SOR.
     MOVE 20 TO CODE-SOR.
     WRITE REC-SOR.


 ECRIRE-TO.
************
     MOVE 2  TO LONG-SOR
     MOVE "TO" TO MOT-SOR.
     MOVE 20 TO CODE-SOR.
     WRITE REC-SOR.



 ECRIRE-SORTIENBRE-WRITE.
**************************
     MOVE 50 TO LONG-SOR
     MOVE "SORTIE" TO MOT-SOR.
     MOVE 7 TO POINTEUR.
     STRING NBRE-WRITE DELIMITED BY SIZE INTO MOT-SOR
     WITH POINTER POINTEUR.
     MOVE POINTEUR TO LONG-SOR.
     MOVE 10 TO CODE-SOR.
     WRITE REC-SOR.


 ECRIRE-EXITPROGRAM.
*********************
     MOVE 12 TO LONG-SOR
     MOVE "EXIT PROGRAM" TO MOT-SOR.
     MOVE 20 TO CODE-SOR.
     WRITE REC-SOR.

 ECRIRE-QNBRE-WRITE.
*********************
     MOVE 50 TO LONG-SOR
     MOVE "Q" TO MOT-SOR.
     MOVE 2 TO POINTEUR.
     STRING NBRE-WRITE DELIMITED BY SIZE INTO MOT-SOR
     WITH POINTER POINTEUR.
     MOVE POINTEUR TO LONG-SOR.
     MOVE 10 TO CODE-SOR.
     WRITE REC-SOR.


 ECRIRE-MOVE1TOEOF.
********************
     MOVE 13 TO LONG-SOR
     MOVE "MOVE 1 TO EOF" TO MOT-SOR.
     MOVE 20 TO CODE-SOR.
     WRITE REC-SOR.
```

```
 ECRIRE-DERNIERESORTIE.
************************
     MOVE 13 TO LONG-SOR
     MOVE "DERNIRESORTIE" TO MOT-SOR.
     MOVE 10 TO CODE-SOR.
     WRITE REC-SOR.



 ECRIRE-FINPROGRAM.
********************
     MOVE 10 TO LONG-SOR
     MOVE "FINPROGRAM" TO MOT-SOR.
     MOVE 10 TO CODE-SOR.
     WRITE REC-SOR.

*   ==============================================================  *
* =================  P R I M I T I V E S  POUR LA 1ERE PASSE =====  *
* ================================================================  *

 LIRE-SOR.
**********
     READ FICHIER-SORTIE AT END MOVE 1 TO FIN-SOR.

 ECR-LIRE.
**********

     PERFORM ECR-MOT.
     PERFORM LIRE-SOR.

 ECR-MOT.
************
     WRITE REC-INV FROM REC-SOR.

*
*
*
*      E C R I T U R E   D U   C O D E  POUR LA 1ERE PASSE
*
*
*


 ECR-USINGEOF.
**************
     MOVE 9  TO LONG-INV
     MOVE "USING EOF" TO MOT-INV.
     MOVE 20 TO CODE-INV.
     WRITE REC-INV.

 ECR-REC.
*********
     MOVE 250 TO LONG-INV.
     MOVE 1 TO POINTEUR.
     STRING REC-C DELIMITED BY SIZE INTO MOT-INV
     WITH POINTER POINTEUR.
     MOVE POINTEUR TO LONG-INV.
     MOVE 20 TO CODE-INV.
     WRITE REC-INV.


 ECR-GOTO.
**********
     MOVE 5  TO LONG-INV
```

```
      MOVE "GO TO" TO MOT-INV.
      MOVE 20 TO CODE-INV.
      WRITE REC-INV.

 ECR-QI.
********
      MOVE 50 TO LONG-INV
      MOVE "Q" TO MOT-INV.
      MOVE 2 TO POINTEUR.
      STRING I DELIMITED BY SIZE INTO MOT-INV
      WITH POINTER POINTEUR.
      MOVE POINTEUR TO LONG-INV.
      MOVE 20 TO CODE-INV.
      WRITE REC-INV.


 ECR-DEPENDINGONQS.
********************
      MOVE 15 TO LONG-INV
      MOVE "DEPENDING ON QS" TO MOT-INV.
      MOVE 20 TO CODE-INV.
      WRITE REC-INV.


 ECR-POINT.
************
      MOVE 1 TO LONG-INV
      MOVE "." TO MOT-INV.
      MOVE 1 TO CODE-INV.
      WRITE REC-INV.


 ECR-Q1.
********
      MOVE 3  TO LONG-INV
      MOVE "QO1" TO MOT-INV.
      MOVE 10 TO CODE-INV.
      WRITE REC-INV.


*
*   ╔═══════════════════════════════════════════════════════════╗
*   ║                    FIN DU PROGRAMME                       ║
*   ║          INVERSION DE P1 PAR RAPPORT A P2                 ║
*   ║                  MODIFICATIONS DE P1                      ║
*   ║                                                           ║
*   ╚═══════════════════════════════════════════════════════════╝
```

```
*
*
*       INVERSION DE P1 PAR RAPPORT A P2
*       MODIFIE LE PROGRAMME P2
*
*
 IDENTIFICATION DIVISION.
**************************
 PROGRAM-ID.  INVP2.

* inversion de P1 par rapport a P2.
* modifications de p2

 ENVIRONMENT DIVISION.
***********************
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
***************
 SELECT FICHTEMP  ASSIGN TO "TEMPO.XXX"
        ORGANIZATION IS SEQUENTIAL.

 SELECT FICHIER-ENTREE  ASSIGN TO FICHIER-SOURCE
        ORGANIZATION IS SEQUENTIAL.

 SELECT FICHIER-SORTIE ASSIGN TO FICHIER-RESULTAT
        ORGANIZATION IS SEQUENTIAL.


 DATA DIVISION.
****************

 FILE SECTION.
***************
 FD FICHTEMP LABEL RECORD IS STANDARD.
 01 REC-TEM.
    02 CODE-TEM       PIC 99.
    02 LONG-TEM       PIC 999.
    02 MOT-TEM.
       03 CAR-TEM PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-TEM.

 FD FICHIER-ENTREE LABEL RECORD IS STANDARD.
 01 REC-ENT.
    02 CODE-ENT       PIC 99.
    02 LONG-ENT       PIC 999.
    02 MOT-ENT.
       03 CAR-ENT PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-ENT.

 FD FICHIER-SORTIE LABEL RECORD IS STANDARD.
 01 REC-SOR.
    02 CODE-SOR       PIC 99.
    02 LONG-SOR       PIC 999.
    02 MOT-SOR.
       03 CAR-SOR PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-SOR.


 WORKING-STORAGE SECTION.
**************************


 01 MOT-GARDE.
    02 CODE-GARDE     PIC 99.
    02 LONG-GARDE     PIC 999.
```

```
    02 MOT-GAR.
       03 CAR-GAR PIC X OCCURS 1 TO 255 TIMES
                        DEPENDING ON LONG-GARDE.

 01 FICHIER-SOURCE        PIC X(12).
 01 FICHIER-RESULTAT      PIC X(12).
 01 NOM-RESULTAT          PIC X(12).
 01 NOM-P1                PIC X(12).
 01 NOM-P1-1              PIC X(12).

 01 P1                PIC X(30).
 01 P2                PIC X(30).
 01 FICH-C            PIC X(30).
 01 REC-C             PIC X(30).
 01 REC1              PIC X(30).

 01 POINTEUR          PIC 9(3).


 01 ETAT-PRESENT    PIC 99.

 01 TROUVE-FICHC    PIC 9.
 01 TROUVE-AT-END   PIC 9.

 01 FIN-FICHIER     PIC 9.
 01 FIN-FICHTEMP    PIC 9.


 PROCEDURE DIVISION.
**********************
*=============================================================
*=============================================================
*============== SAISIE DES PARAMETRES P1, P2, FICHIER-RESULTAT,
*==============                                 FICHC
*=============================================================

      DISPLAY "INVERSION DE P1 % P2, MODIF P2" UPON CONSOLE.

      DISPLAY "entrez le fichier P2 (.MOT)" UPON CONSOLE.
      ACCEPT FICHIER-SOURCE FROM CONSOLE.
      INSPECT FICHIER-SOURCE REPLACING FIRST "    " BY ".MOT".
      DISPLAY "entrez le fichier resultat (.MOT)" UPON CONSOLE.
      ACCEPT FICHIER-RESULTAT FROM CONSOLE.
      MOVE FICHIER-RESULTAT TO NOM-RESULTAT.
      INSPECT FICHIER-RESULTAT REPLACING FIRST "    " BY ".MOT".

      DISPLAY "le fichier source P2 est " FICHIER-SOURCE
              "et le fichier resultat P2 est " FICHIER-RESULTAT
                                               UPON CONSOLE.

      DISPLAY "entrez le fichier P1 " UPON CONSOLE.
      ACCEPT NOM-P1 FROM CONSOLE.
      MOVE NOM-P1 TO NOM-P1-1
      INSPECT NOM-P1-1 REPLACING FIRST "  " BY "-1".


      DISPLAY "entrez le nom du fichier intermédiaire "
                                 UPON CONSOLE.
      ACCEPT FICH-C FROM CONSOLE.

      OPEN INPUT FICHIER-ENTREE
      OPEN OUTPUT FICHIER-SORTIE
      OPEN OUTPUT FICHTEMP.

*=============================================================
*=============================================================
```

```
*================ TRAITEMENT DE P2 ================
*
*
*  modification du nom du programme
*
      PERFORM  LIRE-SUIVANT.
      PERFORM ECRIRE-LIRE UNTIL (MOT-ENT = "PROGRAM-ID").
      PERFORM ECRIRE-LIRE.
      PERFORM ECRIRE-LIRE.
* on a lu le nom du programme et on va le remplacer par le nouveau
      MOVE 10 TO LONG-ENT
      MOVE 1 TO POINTEUR.
      STRING NOM-RESULTAT DELIMITED BY SIZE INTO MOT-ENT
      WITH POINTER POINTEUR.
      MOVE POINTEUR TO LONG-ENT.
      MOVE 20 TO CODE-ENT.
*
*  suppression du SELECT FICH-C
*
      PERFORM ECRIRE-LIRE UNTIL (MOT-ENT = "FILE-CONTROL").
      MOVE 0 TO TROUVE-FICHC.
      PERFORM TRAIT-SELECT UNTIL (TROUVE-FICHC = 1).
*
*  suppression de FD FICHC et
*  sauvetage de la declaretion de REC
*
      PERFORM ECRIRE-LIRE UNTIL (MOT-ENT = "FILE").
      PERFORM ECRIRE-LIRE.
      MOVE 0 TO TROUVE-FICHC.
      PERFORM TRAIT-FD UNTIL (TROUVE-FICHC = 1).
      PERFORM ECRIRE-LIRE.
*
*  ajout de la declaration de REC en WSS
*
* ici, on a ecrit WORKING-STORAGE SECTION. et lu le mot suivant
      CLOSE FICHTEMP
      OPEN INPUT FICHTEMP.
      PERFORM RECOPIER-FICHTEMP.
      PERFORM ECRIRE-01EOFPIC9.
      PERFORM ECRIRE-POINT.
*
*  modifications de la PROCEDURE DIVISION
*
      PERFORM ECRIRE-LIRE UNTIL (MOT-ENT = "PROCEDURE")
      PERFORM ECRIRE-LIRE.
      PERFORM TRAIT-PROC-DIV UNTIL (FIN-FICHIER = 1).

      CLOSE FICHIER-ENTREE
      CLOSE FICHIER-SORTIE
      CLOSE FICHTEMP.

      STOP RUN.
*================================================*
*================================================*
```

```
 TRAIT-SELECT.
***************
      PERFORM ECRIRE-LIRE UNTIL (CODE-ENT = 40).
      PERFORM LIRE-SUIVANT.
      IF (MOT-ENT = FICH-C)
         PERFORM LIRE-SUIVANT UNTIL (CODE-ENT = 1)
         PERFORM LIRE-SUIVANT
         MOVE 1 TO TROUVE-FICHC
      ELSE
         PERFORM ECRIRE-SELECT.

* ═══════════════════════════════════════════════════════════════ *

 TRAIT-FD.
***********
      PERFORM ECRIRE-LIRE UNTIL (CODE-ENT = 41).                    SELECT
      PERFORM LIRE-SUIVANT.
      IF (MOT-ENT = FICH-C)
         PERFORM LIRE-SUIVANT UNTIL (CODE-ENT = 1)                  POINT
         MOVE 1 TO TROUVE-FICHC
*             ici, il faut identifier REC

         PERFORM LIRE-SUIVANT
         PERFORM ECRIRE-LIRE-FICHTEMP UNTIL (CODE-ENT = 41          SELECT
                                 OR MOT-ENT = "WORKING-STORAGE")
         IF (MOT-ENT = "WORKING-STORAGE")
            PERFORM ECRIRE-LIRE
            PERFORM ECRIRE-LIRE
         ELSE
            PERFORM ECRIRE-LIRE UNTIL (MOT-ENT =
                                       "WORKING-STORAGE")
            PERFORM ECRIRE-LIRE
            PERFORM ECRIRE-LIRE

      ELSE
         PERFORM ECRIRE-SELECT.


* ═══════════════════════════════════════════════════════════════ *

 TRAIT-PROC-DIV.
*****************
*
*  ╔═══════════════════════════════════════╗
*  ║ traitement de la PROCEDURE DIVISION   ║
*  ╚═══════════════════════════════════════╝
*


      IF (MOT-ENT = "OPEN" )
         PERFORM TRAIT-OPEN
      ELSE
      IF (MOT-ENT = "READ" )
         PERFORM TRAIT-READ
      ELSE
      IF (MOT-ENT = "CLOSE" )
         PERFORM TRAIT-CLOSE.
      PERFORM ECRIRE-MEME-MOT.
      PERFORM LIRE-SUIVANT.


* ═══════════════════════════════════════════════════════════════ *

 TRAIT-OPEN.
*************
*
```

```
* si OPEN INPUT FICH-C, supprimer cette instruction
* sinon, la recopier
*
      PERFORM GARDER-FICHTEMP
      PERFORM LIRE-SUIVANT
      IF MOT-ENT = "INPUT"
         PERFORM GARDER-FICHTEMP
         PERFORM LIRE-SUIVANT
         IF MOT-ENT = FICH-C
            PERFORM VIDER-FICHTEMP
            PERFORM LIRE-SUIVANT
            IF MOT-ENT = "."
               PERFORM LIRE-SUIVANT
            ELSE
               NEXT SENTENCE
         ELSE
            PERFORM RECOPIER-FICHTEMP
      ELSE
         PERFORM RECOPIER-FICHTEMP.

* ================================================================ *

 TRAIT-READ.
************
*
*  si READ FICH-C, appel à TRAIT-READ-FICHC
*  sinon, recopier les instructions
*
      PERFORM GARDER-FICHTEMP
      PERFORM LIRE-SUIVANT
      IF MOT-ENT = FICH-C
         PERFORM VIDER-FICHTEMP
         PERFORM TRAIT-READ-FICH-C
      ELSE
         PERFORM RECOPIER-FICHTEMP.


* ================================================================ *


 TRAIT-CLOSE.
************
*
*  si CLOSE FICH-C, appel à TRAIT-CLOSE-FICHC
*  sinon, recopier les instructions
*
      PERFORM GARDER-FICHTEMP
      PERFORM LIRE-SUIVANT
      IF MOT-ENT = FICH-C
         PERFORM VIDER-FICHTEMP
         PERFORM TRAIT-CLOSE-FICHC
         PERFORM LIRE-SUIVANT
      ELSE
         PERFORM RECOPIER-FICHTEMP.


* ================================================================ *

 TRAIT-READ-FICH-C.
******************
* EN ENTTRANT ICI, ON A LU : READ FICHC
      PERFORM LIRE-SUIVANT                                    RECORD
      PERFORM LIRE-SUIVANT
      IF MOT-ENT = "INTO"
         PERFORM LIRE-SUIVANT
```

```
         MOVE MOT-ENT TO  REC1
         PERFORM LIRE-SUIVANT.
      IF MOT-ENT = "AT"
         PERFORM LIRE-SUIVANT
         PERFORM LIRE-SUIVANT
         PERFORM ECRIRE-LIRE-FICHTEMP UNTIL (CODE-ENT = 1 )
         MOVE 1 TO TROUVE-AT-END
       ELSE
         MOVE 0 TO TROUVE-AT-END.

      PERFORM ECRIRE-CALLP1-1USINGEOF
      IF REC1 NOT = SPACES
         PERFORM ECRIRE-REC1
         PERFORM ECRIRE-POINT
      ELSE
         PERFORM ECRIRE-REC
         PERFORM ECRIRE-POINT.

      IF ( TROUVE-AT-END = 1 )
         PERFORM ECRIRE-IFEOFEGAL1
         PERFORM RECOPIER-FICHTEMP.



 TRAIT-CLOSE-FICHC.
********************

      PERFORM ECRIRE-IFEOFEGALO
      PERFORM ECRIRE-CALLP1-1USINGEOF
      PERFORM ECRIRE-REC
      PERFORM ECRIRE-POINT.




*    ================================================================ *
* ================  P R I M I T I V E S  ================================ *
* ================================================================ *
 LIRE-SUIVANT.
***************
      READ FICHIER-ENTREE AT END MOVE 1 TO FIN-FICHIER.

 ECRIRE-MOT.
*************
      WRITE REC-SOR.

 ECRIRE-MEME-MOT.
*************
      WRITE REC-SOR FROM REC-ENT.


 ECRIRE-LIRE.
***************
      PERFORM ECRIRE-MEME-MOT.
      PERFORM LIRE-SUIVANT.


 GARDER-FICHTEMP.
******************
* VA ECRIRE LE MOT LU DANS FICHTEMP

      WRITE REC-TEM FROM REC-ENT.


 ECRIRE-LIRE-FICHTEMP.
***********************
```

```
      PERFORM GARDER-FICHTEMP
      PERFORM LIRE-SUIVANT.

 RECOPIER-FICHTEMP.
********************
* va ecrire le contenu de fichtemp dans le fichier de sortie
* puis réinitialiser fichtemp, en faisant CLOSE FICHTEMP
*  et OPEN OUTPUT FICHTEMP

      CLOSE FICHTEMP.
      OPEN INPUT FICHTEMP.
      MOVE 0 TO FIN-FICHTEMP.
      PERFORM ECRIRE-FICHTEMP UNTIL FIN-FICHTEMP = 1.
      CLOSE FICHTEMP
      OPEN OUTPUT FICHTEMP.


 ECRIRE-FICHTEMP.
******************
      READ FICHTEMP
            AT END MOVE 1 TO FIN-FICHTEMP.
      IF FIN-FICHTEMP = 0
         WRITE REC-SOR FROM REC-TEM.

 VIDER-FICHTEMP.
*****************
      CLOSE FICHTEMP.
      OPEN OUTPUT FICHTEMP.

*
*
*
*          E C R I T U R E   D U   C O D E
*
*
*

 ECRIRE-SELECT.
****************
      MOVE 6 TO LONG-SOR
      MOVE "SELECT" TO MOT-SOR.
      MOVE 10 TO CODE-SOR.
      WRITE REC-SOR.

 ECRIRE-CALLP1-1USINGEOF.
**************************
      MOVE 250 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING "CALL """ DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING NOM-P1-1 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      STRING """ USING EOF " DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      MOVE POINTEUR TO LONG-SOR.
      MOVE 20 TO CODE-SOR.
      WRITE REC-SOR.

 ECRIRE-REC1.
**************
      MOVE 250 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING REC1 DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      MOVE POINTEUR TO LONG-SOR.
      MOVE 20 TO CODE-SOR.
```

```
      WRITE REC-SOR.

 ECRIRE-REC.
*************
      MOVE 250 TO LONG-SOR.
      MOVE 1 TO POINTEUR.
      STRING REC-C DELIMITED BY SIZE INTO MOT-SOR
      WITH POINTER POINTEUR.
      MOVE POINTEUR TO LONG-SOR.
      MOVE 20 TO CODE-SOR.
      WRITE REC-SOR.


 ECRIRE-IFEOFEGAL1.
*********************
      MOVE 13 TO LONG-SOR
      MOVE "IF (EOF = 1) " TO MOT-SOR.
      MOVE 20 TO CODE-SOR.
      WRITE REC-SOR.

 ECRIRE-IFEOFEGALO.
*********************
      MOVE 13 TO LONG-SOR
      MOVE "IF (EOF = 0) " TO MOT-SOR.
      MOVE 20 TO CODE-SOR.
      WRITE REC-SOR.

 ECRIRE-01EOFPIC9.
*******************

      MOVE 13 TO LONG-SOR
      MOVE "01 EOF PIC 9 " TO MOT-SOR.
      MOVE 20 TO CODE-SOR.
      WRITE REC-SOR.


 ECRIRE-POINT.
***************
      MOVE 1 TO LONG-SOR
      MOVE "." TO MOT-SOR.
      MOVE 1 TO CODE-SOR.
      WRITE REC-SOR.

*
*     ╔═══════════════════════════════════════════════════════╗
*     ║                    FIN DU PROGRAMME                   ║
*     ║            INVERSION DE P1 PAR RAPPORT A P2           ║
*     ║                   MODIFICATION DE P2                  ║
*     ║                                                       ║
*     ╚═══════════════════════════════════════════════════════╝
```

BIBLIOGRAPHIE

Comprendre, connaître et maitriser le COBOL,
norme ANSI COBOL 1974
A CLARINVAL

Structured programming
O J DAHL, E W DIJKSTRA, C A R HOARE.
Academic Press : 1972

Principles of program design
M A JACKSON, Academic Press : 1975

The art of computer programming,
vol 1 : fundamental algorithms
Donald E KNUTH, 1972

Conception et réalisation des applications de gestion : une approche basée sur l'explicitation des raisonnements.
Isabelle MATHIEU : FNDP, 1986